Final year project

Esercitazioni di Calcolo Numerico

Studente:
Mirko Mauri

Matricola:
735860

Docenti:
Dott.ssa Cristina Tablino Possio
Dott.ssa Milvia Francesca Rossini

18 giugno 2012

Indice

Parte I
ARITMETICA FLOATING POINT

1 INTRODUZIONE A MATLAB

1.1 Matlab come linguaggio di programmazione

Quadrati Magici

Calcolare il rango dei quadrati magici di dim 3:24: v é il vettore dei ranghi dei quadrati magici ed e é il vettore delle dimensioni del ker dei quadrati magici

```
v = 3:24;
for k = 3:24
    v(k-2) = rank(magic(k));
end
n = 3:24;
e = n-v;
A = [n; v; e]';
disp(A);
```

```
     3     3     0
     4     3     1
     5     5     0
     6     5     1
     7     7     0
     8     3     5
     9     9     0
    10     7     3
    11    11     0
    12     3     9
    13    13     0
    14     9     5
    15    15     0
    16     3    13
    17    17     0
    18    11     7
    19    19     0
    20     3    17
    21    21     0
    22    13     9
    23    23     0
    24     3    21
```

Somma dei primi n numeri dispari

```
function  s  = somma( n )
%Somma dei primi n numeri naturali dispari
%METODO SENZA CICLO FOR
sommaSenzaFor = sum([1:2:n]);
%METODO CON CICLO FOR
sommaConFor = 0;
for k = 1:2:n
    sommaConFor = sommaConFor+k;
end
s =[sommaSenzaFor, sommaConFor];
```

esercizio 1

```
A = [2 -1 0; 0 -2 1];
B = [4 1 0; 0 1 4];
b = [6 0 1];
u = [4; 9; -3];
v = [1; 7; -3];

% calcolare C come prodotto di A e B'
C = A*(B');
disp('Matrice C = A*B :');
disp(C);
% calcolare D come prodotto termine a termine di A e B
disp('Matrice D = A.*B :');
D = A.*B;
disp(D);
% calcolare il prodotto scalare tra i vettori riga di B
f = B(1,:)*B(2,:)';
g = sum(B(1,:).*B(2,:));
disp('Prodotto scalare tra le righe di B :');
disp('METODO1: prodotto matriciale');
disp(f);
disp('METODO1: prodotto termine a termine');
disp(g);
if(f==g)
    disp('OSS. I due metodi sono equivalenti');
end
% calcolare e come prodotto di A per il vettore b
e = A*(b');
disp('Matrice e = A*b :');
disp(e);
% dividere termine a termine u per v
format rat;
```

```
disp('Matrice d = u./v :');
d = u./v;
disp(d);
```

```
Matrice C = A*B :
     7    -1
    -2     2

Matrice D = A.*B :
     8    -1     0
     0    -2     4

Prodotto scalare tra le righe di B :
METODO1: prodotto matriciale
     1

METODO1: prodotto termine a termine
     1

OSS. I due metodi sono equivalenti
Matrice e = A*b :
    12
     1

Matrice d = u./v :
       4
       9/7
       1
```

1.2 Plots and Plotting Tools

esercizio 3

Disegnare il grafico delle funzioni $y = \sin(kx)$ per indicati valori di k

```
k = [1/3,1/2,1,2];
for n = [1:4];
    x = [-pi:0.05:pi];
    y(n,:) = sin(k(n)*x)
end
figure
plot(x,y(1,:),':k', x,y(2,:),'--b',x,y(3,:),'-r',x,y(4,:),'-.m');
legend('sin(1/3*x)','sin(1/2*x)','sin(x)','sin(2*x)');
axis([-pi pi -1.2 1.2]);
xlabel('x'); ylabel('y');
```

esercizio 4

L'utente sceglie di disegnare una delle tre funzioni:

1. $y = x^3 - 3x \quad per \quad -3 < x < 3$
2. $y = 3x\cos(2x) \quad per \quad 0 < x < 2\pi$
3. $y = x^3 - 3x \quad per \quad -8\pi < x < 8\pi$

```
disp('La seguente funzione permette di visualizzare')
disp('uno dei tre grafici seguenti:')
disp('1. y = x^3-3*x per -3 < x < 3')
disp('2. y = 3x*cos(2x) per 0 < x < 2*pi')
disp('3. y = x^3-3*x per -8*pi < x < 8*pi')
scelta = input('Scegliere un intero tra 1,2,3: ');
switch scelta
    case 1
        x = [-3:0.05:3];
        y = x.^3-3*x;
        figure
        plot(x,y,'r')
        grid on
        xlabel('x'); ylabel('y');
        title('Grafico della funzione 1');
        legend('y=x^3-3*x');
    case 2
        x = [0:0.05:2*pi];
        y = 3*x.*cos(2*x);
        figure
        plot(x,y,'b')
        grid on
        xlabel('x'); ylabel('y');
        title('Grafico della funzione 2');
        legend('y=3x*cos(2x)');
    case 3
        x = [-8*pi:0.05:8*pi];
        y = sin(x)./x;
        figure
        plot(x,y,'m')
        grid on
        xlabel('x'); ylabel('y');
        title('Grafico della funzione 3');
        legend('y=sin(x)./x');
    otherwise
        disp('Nessuna visualizzazione permessa');
        return;
end
```

1.3 Problema $3n + 1$

L'utente sceglie di disegnare una delle tre funzioni:

- per n=1 stop
- per n pari, sostituisce n con n/2
- per n dispari, sostituisce n con 3n+1

```
function y = threenplus1( n )
y=n;i=0;
while n~=1
    i=i+1;
    if rem(n,2)==0
        n=n/2;
    else
        n=3*n+1;
    end
    y=[y; n];
end
figure
semilogy([0:i],y,'.-');
title('problema 3n+1: # elementi sequenza');
```

Decidiamo di valutare la funzione in 7

```
     7     22     11     34     17     52     26
    13     40     20     10      5     16      8
     4      2      1
```

Rappresentare il numero di elementi della sequenza 3n+1 generata da un k compreso tra 1 e 1000

```
x = [1:1000];
L = zeros(1,1000);
MaxL = 0;
MaxK = 0;
for k = [1:1000]
    tmp = size(threenplus1(k)');
    L(k) = tmp(1,2);
    if(L(k)>MaxL)
        MaxL = L(k);
        MaxK = k;
    end
```

```
end
disp('Il numero massimo di elementi della sequenza generata da k :');
disp(MaxL);
disp('Il valore di k in corrispondenza del max: ');
disp(MaxK);
figure
semilogy(x,L,'.');
title('Numero di elementi della sequenza 3n+1 generata da k');
ylabel('numero di elementi della sequenza');
```

```
Il numero massimo di elementi della sequenza generata da k :
     179

Il valore di k in corrispondenza del max:
     871
```

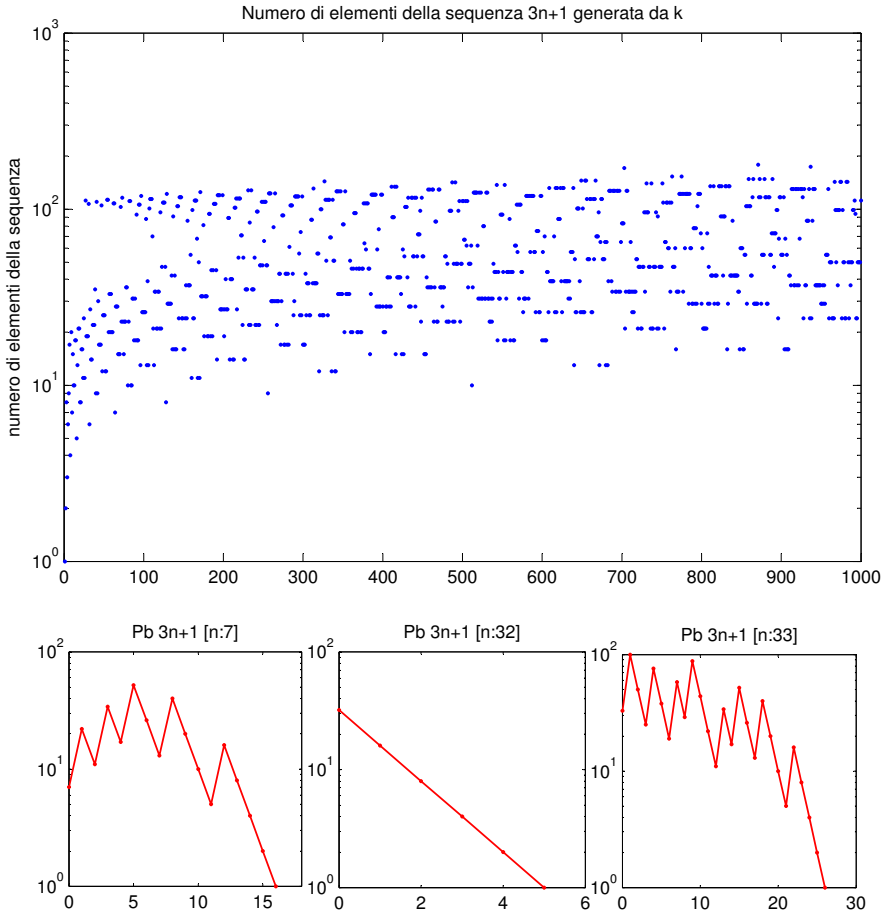

Figura 1: `threenplusGrafico` e `threenplus(n)`

1.4 Sequenza di Fibonacci

Metodo iterativo per il calcolo dei numeri della sequenza di Fibonacci

```
function f = fibonacci(n)
f = zeros(n,1);
f(1)=1;
f(2)=2;
for k = 3:n
    f(k) = f(k-1)+f(k-2);
end
y = f(2:n)./f(1:n-1);
figure
plot([1:n-1],y);
title('Valutazione del rapporto tra due numeri successivi ...
```

Metodo ricorsivo per il calcolo dei numeri della sequenza di Fibonacci

```
function f = fibonacciConRicorsione( n )
if(n==0)
    f=1;
    return;
elseif(n==1)
    f=1;
    return;
else
    f = fibonacciConRicorsione(n-1)+fibonacciConRicorsione(n-2);
end
```

Costo computazionale dei due algoritmi descritti

```
n=30;
x = [1:n];
y1 = zeros(1,n);
y2 = zeros(1,n);
for k = [1:n]
    tic;
    fibonacci(k);
    y1(k)= toc;
    tic;
    fibonacciConRicorsione(k);
    y2(k)= toc;
end
figure
semilogy(x,y1,'r',x,y2,'b');
title('Costo computazionale: sequenza di fibonacci');
legend('metodo iterativo','metodo ricorsivo');
```

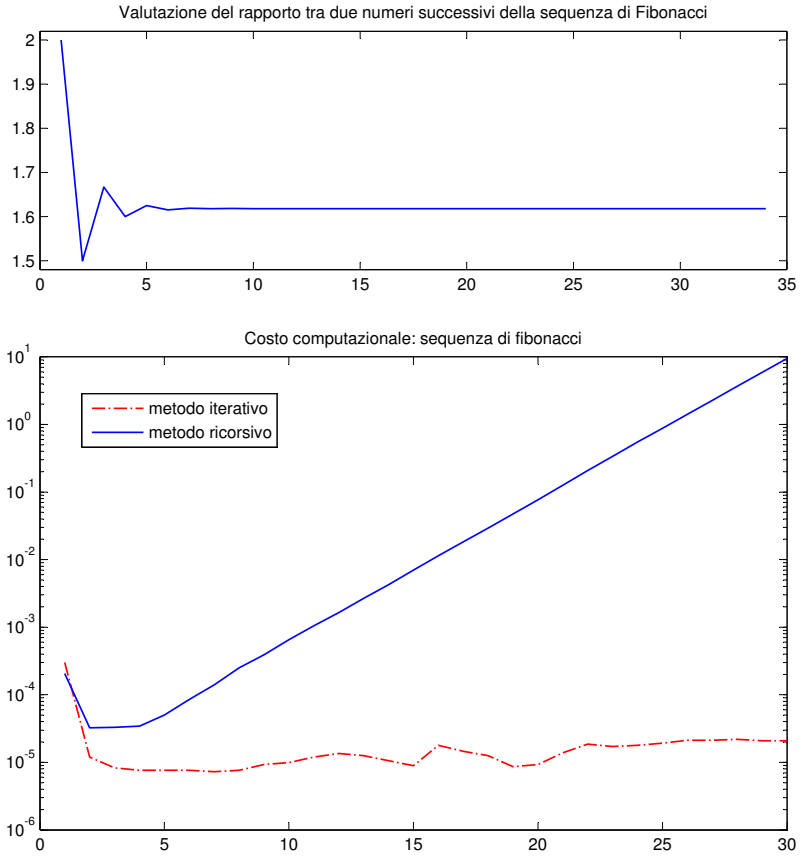

Figura 2: *costo comput* $_{iterativo}$ $<<$ *costo comput* $_{ricorsivo}$

2 SINGOLA PRECISIONE

2.1 Singola e Doppia Precisione

```
format long e
disp('eps,realmax,realmin in singola precisione')
disp([eps('single') realmax('single') realmin('single')]')
p = single(pi)
p1 = double(p)
disp('nota la differenza da pi:')
disp(pi)
```

```
eps,realmax,realmin in singola precisione
   1.1920929e-007
   3.4028235e+038
   1.1754944e-038


p =

   3.1415927e+000


p1 =

    3.141592741012573e+000

nota la differenza da pi:
    3.141592653589793e+000
```

Nota il passaggio dalla singola precisione alla doppia precisione può causare una perdita di precisione (e.g. conversione dalla base 10 alla base 2)

Eseguire la differenza nell'aritmetica a singola precisione e confrontare con il risultato in doppia precisione

```
x =.147554326;
y =.147551742;
sumSingle = single(x)-single(y);
sumDouble = x-y;
sum = .000002584; % differenza in aritmetica esatta
disp([sumSingle sumDouble]);
errS = abs(sumSingle - sum)/sum;
```

```
amplS = errS/(1/2*eps('single'));
errD = abs(sumDouble - sum)/sum;
amplD = errD/(1/2*eps);
disp([x y]')
disp('floating x y')
disp([single(x) single(y)]')
disp('singola precisione  doppia precisione')
disp([sumSingle sumDouble])
disp('errore relativo coefficiente di amplificazione - Single -')
disp([errS amplS])
disp('errore relativo coefficiente di amplificazione - Double -')
disp([errD amplD])
```

Osservazione

Errori di conversione dalla base 10 alla base 2 e viceversa nel floating di x e y. Tuttavia, l'obiettivo dell'esercizio è di valutare l'ordine delle grandezze in gioco

```
   2.5779009e-006   2.5840000e-006

    1.475543260000000e-001
    1.475517420000000e-001

floating x y
   1.4755432e-001
   1.4755175e-001

singola precisione  doppia precisione
   2.5779009e-006   2.5840000e-006

errore relativo     coefficiente di amplificazione   - Single -
   2.3603381e-003   3.9599902e+004

errore relativo     coefficiente di amplificazione   - Double    -
    3.114091717846692e-014    2.804924460018382e+002
```

3 ESERCITAZIONE 3

3.1 *esercizio 1*

Mostriamo che l'aritmetica floating point non conserva le usuali proprietà delle operazioni elementari: l'insieme dei numeri macchina $\mathfrak{F}$ non è un campo rispetto le operazioni di somma e prodotto.

(proprietà associativa della somma)

$$\begin{aligned} a &= .11e + 0 \\ b &= .13e - 1 \\ c &= .14e - 1 \\ a + b &= .123e + 0 \approx .12e + 0 \\ b + c &= .27e - 1 \approx .27e - 1 \\ (a + b) + c &= .134e - 1 \approx .13e + 0 \\ a + (b + c) &= .137e - 1 \approx .14e + 0 \\ (a + b) + c &\neq a + (b + c) \end{aligned}$$

(proprietà associativa del prodotto)

$$\begin{aligned} a &= .11e + 1 \\ b &= .31e + 1 \\ c &= .25e + 1 \\ ab &= .341e + 1 \approx .34e + 1 \\ bc &= .775e - 1 \approx .78e + 1 \\ (ab)c &= .85e - 1 \approx .85e + 1 \\ a(bc) &= .858e - 1 \approx .86e + 1 \\ (ab)c &\neq a(bc) \end{aligned}$$

(proprietà distributiva del prodotto rispetto alla somma)

$$\begin{aligned} a &= .11e + 1 \\ b &= .23e + 1 \\ c &= .24e + 1 \\ ab &= .253e + 1 \approx .25e + 1 \\ bc &= .264e - 1 \approx .26e + 1 \\ ab + ac &= .51e - 1 \approx .51e + 1 \\ b + c &= .47e - 1 \approx .47e - 1 \\ a(b + c) &= .517e - 1 \approx .52e + 1 \\ ab + bc &\neq a(b + c) \end{aligned}$$

("debole" proprietà invariantiva della divisione)

$$\begin{aligned} a &= .70e+1 \\ b &= .10e+1 \\ \frac{b}{a} &= .14285e+0 \approx .14e+0 \\ a\frac{b}{a} &= .98e+1 \approx .98e+1 \\ a\frac{b}{a} &\neq b \end{aligned}$$

3.2 *esercizio 2*

$$\begin{aligned} x &= .916e+1 \\ y &= .928e+1 \\ z &= .167e-2 \\ zx &= .152972e+0 \approx .153e+0 \\ zy &= .154976e+0 \approx .155e+0 \\ zx+zy &= .308e+0 \approx .308e+0 \\ x+y &= .1844e+2 \approx .184e+2 \\ z(x+y) &= .30728e+0 \approx .307e+0 \\ zx+zy &\neq z(x+y) \end{aligned}$$

3.3 *esercizio 3*

$$\begin{aligned} x &= .1234567e+0 \\ y &= .6666325e+4 \\ z &= -.6666325e+4 \\ x+y &= .66664484567e+4 \approx .6666448e+4 \\ (x+y)+z &= .1230000e+0 \approx .1230000e+0 \\ x+(y+z) &= x \\ (x+y)+z &\neq x+(y+z) \\ e_{rel} &= \frac{.1234567-.1230000}{.1234567} = .0037 \\ e_p &= 0.37\% \end{aligned}$$

L'ordine con cui sono svolte le operazioni può causare una perdita di accuratezza nel risultato. All'ultimo passaggio la somma algebrica di addendi discordi

amplifica l'errore accumulato. La **cancellazione numerica** è fonte di instabilità numerica

3.4 *esercizio 4*

```
y1(10)=1;
y2(1)=1;
for k = 1:9
    y1(k) = 10^-16;
    y2(k+1)= 10^-16;
end
disp('   sum1')
disp(sum(y1))
disp('   sum2')
disp(sum(y2))
```

```
   sum1
    1.000000000000001e+000

   sum2
     1
```

Sommare addendi dello stesso ordine di grandezza previene la perdita di cifre significative del risultato.

$$1 + \sum_{i=0}^{9} 10^{16} = ((((1) + 10^{16}) + 10^{16}) \dots)$$

$$\sum_{i=0}^{9} 10^{16} + 1 = (((10^{16}) + 10^{16}) \dots) + 1$$

Nel primo caso (sum1) le prime nove somme parziali sono numeri macchina; al decimo passo viene eseguito l'unico arrotondamento dell'algoritmo. Nel secondo caso (sum2) il termine n-esimo della serie é minore dello spacing del segmento cui appartiene la somma parziale n-esima, il che introduce un errore di arrotondamento ad ogni passo.

3.5 *esercizio 5*

```
format long e
a = 1.234567890123400e+15;
b = -1.234567890123401e+15;
c = 0.06;
somma = -0.94;
```

```
sommaReal = somma*ones(1,3);
sommaAppr = [(a+b)+c  (a+c)+b  a+(b+c)];
err = abs(sommaAppr-sommaReal)./abs(sommaReal);
ampl = err./(1/2*eps);
disp('    a                 b                   c')
disp([a b c])
disp('   (a+b)+c           (a+c)+b              a+(b+c)')
disp([(a+b)+c  (a+c)+b  a+(b+c)])
disp('    errore relativo       coefficiente di amplificazione')
disp([err; ampl]')

a = 0.23371258e-4;
b = 0.33678429e+2;
c = -0.33677911e+2;
somma = 5.41371258e-4;
sommaReal = somma*ones(1,3);
sommaAppr = [(a+b)+c  (a+c)+b  a+(b+c)];
err = abs(sommaAppr-sommaReal)./abs(sommaReal);
ampl = err./(1/2*eps);
disp('    a                 b                   c')
disp([a b c])
disp('   (a+b)+c           (a+c)+b              a+(b+c)')
disp([(a+b)+c  (a+c)+b  a+(b+c)])
disp('    errore relativo       coefficiente di amplificazione')
disp([err; ampl]')
```

```
    a                       b                       c
    1.234567890123400e+015  -1.234567890123401e+015  6.000000000000000e-002

   (a+b)+c                 (a+c)+b                 a+(b+c)
   -9.400000000000000e-001 -1.000000000000000e+000 -1.000000000000000e+000

    errore relativo          coefficiente di amplificazione
                        0                       0
    6.382978723404262e-002   5.749276120047448e+014
    6.382978723404262e-002   5.749276120047448e+014

    a                       b                       c
    2.337125800000000e-005  3.367842900000000e+001  -3.367791100000000e+001

   (a+b)+c                 (a+c)+b                 a+(b+c)
    5.413712580022434e-004   5.413712580022434e-004   5.413712579995743e-004

    errore relativo          coefficiente di amplificazione
    4.143979020229046e-012   3.732564474266937e+004
    4.143979020229046e-012   3.732564474266937e+004
    7.862585363144806e-013   7.081987302325533e+003
```

Osservazione

Sommare addendi dello stesso ordine di grandezza previene la perdita di cifre significative del risultato. Anticipare le somme di addendi discordi consente di contenere l'amplificazione dell'errore accumulato sino a quel passo.

3.6 *esercizio 6*

Calcolare le soluzioni dell'equazione $ax^2 + bx + c = 0$

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$
$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

L'algoritmo usualmente utilizzato per il calcolo delle radici di un polinomio di secondo grado è in generale instabile in aritmetica floating point. Le somme algebriche sono fonte di cancellazione numerica. Nel calcolo di x_1 ovvero di x_2, $-b \pm \sqrt{\Delta}$ è somma algebrica di termini discordi. Per ovviare eventuali errori dovuti a cancellazione numerica, occorre dunque razionalizzare o ricordare che il prodotto delle radici di un polinomio a coefficienti in un campo (algebricamente chiuso o sul quale esso si fattorizzi in fattori lineari) è uguale al termine noto del polinomio, nel nostro caso c. In entrambi i casi si definisce un algoritmo (più stabile):

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$
$$x_2 = \frac{c}{x_1}$$

Resta da valutare il calcolo di Δ. Occorre dunque aumentare il numero di cifre significative oppure, salvo aver implementato un algoritmo sufficientemente stabile per il calcolo della radice quadrata, calcolare:

$$\Delta = b^2 - 4ac = (b + 2\sqrt{ac})(b - 2\sqrt{ac})$$

3.7 *esercizio 7*

Studiare la stabilità degli algoritmi seguenti

$$fl(fl(a^2) - fl(b^2))$$
$$fl(fl(a - b)fl(a + b))$$

Si mostra che l'errore analitico dei due algoritmi è

$$E_{AN,1} < (1 + \frac{x^2 + y^2}{x^2 - y^2})u$$
$$E_{AN,2} < 3u$$

L'errore analitico del secondo algoritmo è controllato dalla maggiorazione dell'errore del primo per

$$3u < (1 + \frac{x^2 + y^2}{|x^2 - y^2|})u$$
$$\{(x, y) \in \Re^2 | -\sqrt{3}x < y < -\frac{1}{\sqrt{3}}x \vee \frac{1}{\sqrt{3}}x < y < \sqrt{3}x\}$$

```
a = input('inserire il numero a: ');
b = input('inserire il numero b: ');
result = a^2-b^2;
digits(5);
disp('numeri macchina: a b')
disp(vpa([a b],5))
u = 1/2*10^(-4);
result1 = vpa(vpa(a^2,5)-vpa(b^2,5),5);
result2 = vpa(vpa(a-b,5)*vpa(a+b,5),5);
disp('   result1       result2')
disp([result1   result2])
err1 = abs(result1-result)/abs(result);
err2 = abs(result2-result)/abs(result);
disp('   err1       err2')
disp([err1   err2])
stimaErr1 = 1+(a^2+b^2)/abs((a^2-b^2)*u);
stimaErr2 = 3*u;
disp('   maggiorazione errore: err1       err2')
disp([stimaErr1   stimaErr2])
disp('   coefficiente di amplificazione: err1    err2')
disp([err1/u err2/u])
disp('   maggiorazione coefficienti di amplificazione: err1 err2')
disp([stimaErr1/u stimaErr2/u])
```

```
inserire il numero a: 666893452
inserire il numero b: 666892452
numeri macchina: a b
[ 6.6689e8, 6.6689e8]

   result1       result2
[ 1.3338e12, 1.3338e12]
```

```
   err1       err2
[ 1.1612e-8, 0]

   maggiorazione errore: err1       err2
    1.333785904100750e+010    1.500000000000000e-004

   coefficiente di amplificazione: err1    err2
[ 0.00023224, 0]

   maggiorazione coefficienti di amplificazione: err1    err2
    2.667571808201499e+014    3.000000000000000e+000
```

4 STIMA DELL'ESPONENZIALE

4.1 Stima dell'esponenziale: $\lim_{x\to+\infty}(1+\frac{1}{n})^n = e$

Stimare e con la seguente funzione : $f(n) = (1+\frac{1}{n})^n$

```
f = inline('(1+1/x)^x');
x = 10.^[0:16];
f = vectorize(f);
y1 = f(x);
y2 = exp(1)*ones(1,size(x,2));
err = (y1-y2)./y2;
format long e
disp('    stima di e     errore relativo')
disp([y1' err'])
disp('nota la differenza da e: ')
disp(exp(1))
```

```
    stima di e                errore relativo
    2.000000000000000e+000   -2.642411176571154e-001
    2.593742460100002e+000   -4.581547323576922e-002
    2.704813829421529e+000   -4.954599959619296e-003
    2.716923932235594e+000   -4.995421038523148e-004
    2.718145926824926e+000   -4.999541721436175e-005
    2.718268237192298e+000   -4.999947616083742e-006
    2.718280469095753e+000   -5.000818082661048e-007
    2.718281694132082e+000   -4.941612836756667e-008
    2.718281798347358e+000   -1.107747088422168e-008
    2.718282052011560e+000    8.224037418842151e-008
    2.718282053234788e+000    8.269037436295259e-008
    2.718282053357110e+000    8.273537433139431e-008
    2.718523496037238e+000    8.890453361460873e-005
    2.716110034086901e+000   -7.989584999638583e-004
    2.716110034087023e+000   -7.989584999190946e-004
    3.035035206549262e+000    1.165270557209954e-001
    1.000000000000000e+000   -6.321205588285577e-001

nota la differenza da e:
    2.718281828459046e+000
```

Visualizziamo i risultati:

```
x = 10.^[0:0.05:16];
y1 = f(x);
```

```
y2 = exp(1)*ones(1,size(x,2));
figure;
semilogx(x,y1,'.-',x,y2,'r-');
axis([min(x) max(x)  2 2.8]);
figure;
err = (y1-y2)./y2;
semilogx(x,err,'r');
```

4.2 Stima dell'esponenziale: $e^x = \sum_{i=0}^{+\infty} \frac{x^n}{n!}$

Confrontare la stima di f col polinomio di Taylor dell'esponenziale di ordine n

```
g = inline('1/factorial(n)');
k = 20;
x = [0:k-1];
y1 = zeros(1,k);
y2 = exp(1)*ones(size(y1));
err = y1;
g = vectorize(g);
y1(1) = 2;
err(1) = (y1(1)-y2(1))/y2(1);
for i = [2:k]
    y1(i) = y1(i-1)+g(i);
    err(i) = (y1(i)-y2(i))/y2(i);
end
err = (y1-y2)./y2;
disp('    stima di e         errore relativo')
disp([y1' err'])
disp('nota la differenza da e: ')
disp(exp(1))
figure;
plot(x,y1,'.-',x,y2,'r-');
axis([min(x) max(x)  2 2.8]);
figure;
plot(x,err,'r');
axis([min(x) max(x)  -0.2 0])
```

Osservazione

L'annullamento dell'errore relativo non é indice del calcolo di un risultato esatto: le cifre infinite di e non hanno rappresentazione in nessuna aritmetica finita. Ad ogni passo della ricorsione l'errore é abbattuto di un ordine di grandezza fino al diciassettesimo passo. A tale passo il termine n-esimo della serie é minore dello

spacing del segmento cui appartiene e:

$$fl(\sum_{i=0}^{k}\frac{1}{n!}+\frac{1}{(k+1)!}) = fl(\sum_{i=0}^{k}\frac{1}{n!}) \quad per\ k > 16$$

```
spacing(e):  4.440892098500626e-016
```

stima di e	errore relativo	termine n-esimo
2.000000000000000e+000	-2.642411176571154e-001	5.000000000000000e-001
2.500000000000000e+000	-8.030139707139430e-002	1.666666666666667e-001
2.666666666666667e+000	-1.898815687615397e-002	4.166666666666666e-002
2.708333333333333e+000	-3.659846827343931e-003	8.333333333333333e-003
2.716666666666666e+000	-5.941848175819229e-004	1.388888888888889e-003
2.718055555555555e+000	-8.324114928817323e-005	1.984126984126984e-004
2.718253968253968e+000	-1.024919667471039e-005	2.480158730158730e-005
2.718278769841270e+000	-1.125202598007116e-006	2.755731922398589e-006
2.718281525573192e+000	-1.114254784460283e-007	2.755731922398589e-007
2.718281801146385e+000	-1.004776647358234e-008	2.505210838544172e-008
2.718281826198493e+000	-8.316108397236229e-010	2.087675698786810e-009
2.718281828286169e+000	-6.359782939257374e-011	1.605904383682161e-010
2.718281828446759e+000	-4.519830118820566e-012	1.147074559772973e-011
2.718281828458230e+000	-2.999496890824318e-013	7.647163731819816e-013
2.718281828458995e+000	-1.862432709988956e-014	4.779477332387385e-014
2.718281828459043e+000	-9.802277420994503e-016	2.811457254345521e-015
2.718281828459046e+000	0	1.561920696858623e-016
2.718281828459046e+000	0	8.220635246624330e-018
2.718281828459046e+000	0	4.110317623312165e-019
2.718281828459046e+000	0	1.957294106339126e-020

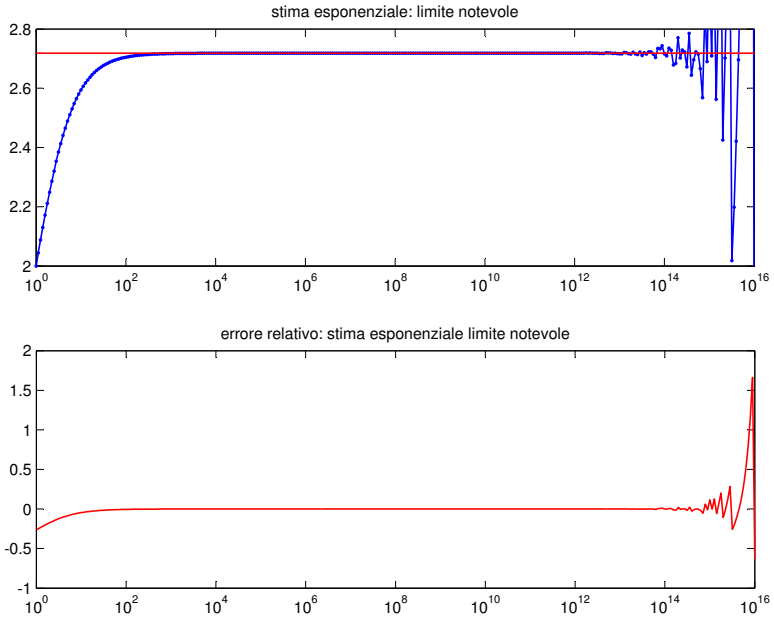

Figura 3: convergenza ed errore relativo: $\lim_{x\to+\infty}(1+\frac{1}{n})^n = e$

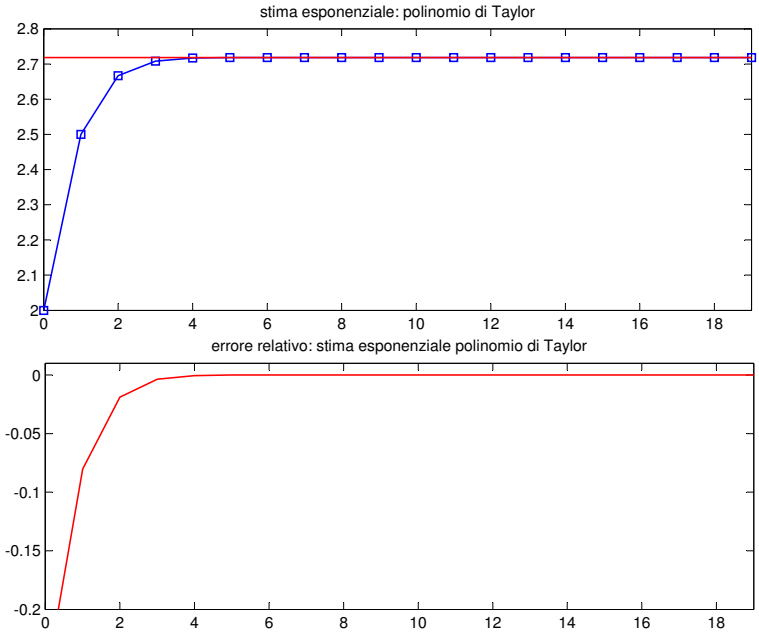

Figura 4: convergenza ed errore relativo: $e^x = \sum_{i=0}^{+\infty} \frac{x^n}{n!}$

5 ALGORITMO DI HORNER

Confrontare l'algoritmo di Horner o delle moltiplicazioni innestate con la valutazione del polinomio fattorizzato per

$$f(x) = x^6 - 6x^5 + 15x^4 - 20x^3 + 15x^2 - 6x + 1 \in Z[x]$$

```
a  = [1 -6 15 -20 15 -6 1];
x  = linspace(0.998, 1.002);
%.............................................................
% ALGORITMO DI VALUTAZIONE (home-made)
tmp = ones(1,length(x));
s = a(1)*tmp;
for i=2:length(a)
    s = s.*x+a(i)*tmp
end
%.............................................................
% FUNCTION MATLAB POLYVAL
y  = polyval(a,x);
figure;
subplot(1,2,1)
plot(x,s,'r');
title('algoritmo di Horner');
subplot(1,2,2)
plot(x,y,'r');
title('funzione MATLAB POLYVAL');
%.............................................................
% ALGORITMO DI VALUTAZIONE DEL POLINOMIO FATTORIZZATO
y1 = (x-1).^6;
figure;
plot(x,y1);
figure;
plot(x,y,'r',x,y1,'b');
legend('Horner', '(x-1).^6')
```

Osservazione

L'errore algoritmico é dovuto a eventuale cancellazione numerica o alla differenza degli ordini di grandezza dei coefficienti del polinomio.

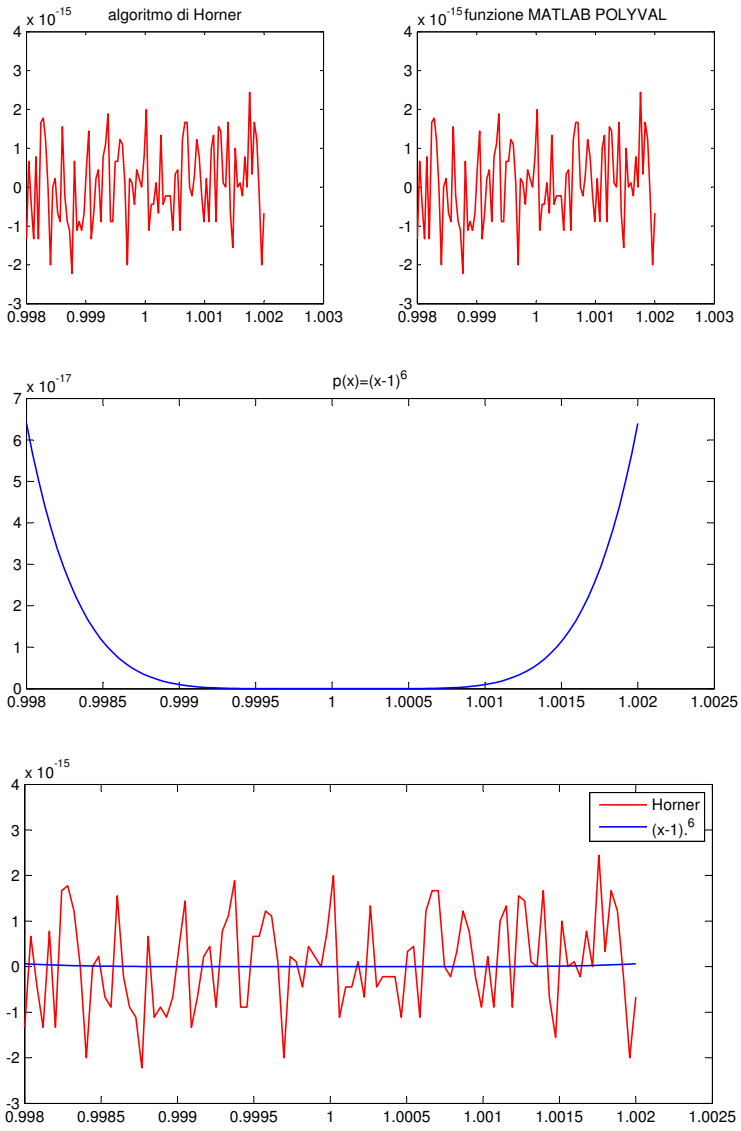

Figura 5: stabilità dell' algoritmo di Horner: $p(x) = (x-1)^6$

6 ESERCITAZIONE 4

$$c_x = \frac{x}{f(x)}\frac{df(x)}{dx}$$
$$c_{x_i} = \frac{x_i}{f(x)}\frac{\partial f(x)}{\partial x_i}$$
$$|E_{IN}| \leq \sum_{i=1}^{n} |\frac{x_i}{f(x)}\frac{\partial f(x)}{\partial x_i}|u$$

6.1 *esercizio 1*

Valutare per quali valori di α il problema del calcolo delle soluzioni dell'equazione $x^2 - 4x + \alpha = 0$ é malcondizionato

$$x_1 = 2 + \sqrt{4-\alpha}$$
$$x_2 = 2 - \sqrt{4-\alpha}$$
$$c_\alpha = \frac{\alpha}{2 \pm \sqrt{4-\alpha}}\left(\frac{\mp 1}{2\sqrt{4-\alpha}}\right)$$

$$x \to \infty \quad |c_\alpha| \to 1/2 \qquad x \to 0 \quad |c_\alpha| \to 1 \qquad x \to 4 \quad |c_\alpha| \to \infty$$

Il problema é malcondizionato in un intorno di 4.

6.2 *esercizio 2*

Studiare l'errore inerente e il condizionamento dei problemi seguenti

$$\phi(x) = x - 1 \qquad c_x = \frac{x}{x-1} \qquad x \to 1 \quad |c_x| \to \infty$$
$$\phi(x) = e^x \qquad c_x = x \qquad x \to \infty \quad |c_x| \to \infty$$
$$\phi(x) = ln(|x|) \qquad c_x = \frac{1}{ln(|x|)} \qquad x \to \pm 1 \quad |c_x| \to \infty$$
$$\phi(x) = \sin(x) \qquad c_x = x\cot(x) \qquad x \to k\pi \quad |c_x| \to \infty$$
$$\phi(a,b,c) = a + b + c \qquad c_a = \frac{a}{a+b+c} \quad a+b+c \approx 0$$
$$c_b = \frac{b}{a+b+c} \quad a+b+c \approx 0$$
$$c_c = \frac{c}{a+b+c} \quad a+b+c \approx 0$$
$$\phi(a,b) = a^2 - b^2 \qquad c_a = \frac{2a^2}{a^2-b^2} \quad a \approx b$$
$$c_b = \frac{-2b^2}{a^2-b^2} \quad a \approx b$$

In particolare

$$\phi(x) = \sqrt{x^2+1} - |x| \qquad c_x = \frac{x}{\sqrt{x^2+1}-|x|}\left(\frac{x}{\sqrt{x^2+1}} - \frac{x}{|x|}\right) = \frac{|x|}{\sqrt{x^2+1}} < 1$$

Il problema é ben condizionato. A fronte di un errore inerente compatibile con l'unità d'arrotondamento, l'errore algoritmico della precedente procedura di calcolo non é controllato per x grande. La cancellazione numerica é fonte d'instabilità per l'algoritmo. Supponiamo ad esempio x sufficientemente grande affinchè $fl(x^2+1) \approx fl(x^2) \Rightarrow \phi(x) = 0 \Rightarrow e_{rel} = 1$. Formalizzando le precedenti osservazioni

$$\phi_1(x) = \sqrt{x^2+1} - |x| \qquad |E_{ALG,1}| \leq \left(1 + \frac{2\sqrt{x^2+1}}{\sqrt{x^2+1}-|x|}\right) u \to \infty \quad x \to \infty$$

Proponiamo dunque un algoritmo stabile, razionalizzando ϕ_1

$$\phi_2(x) = \frac{1}{\sqrt{x^2+1}+|x|} \qquad |E_{ALG,1}| \leq 3u$$

6.3 *esercizio 3*

Verificare che il problema é malcondizionato

$$\begin{cases} 3x + 5y = 10 \\ 3.01x + 5.01y = 1 \end{cases}$$

Per piccole perturbazioni dei dati iniziali le soluzioni subiscono grandi perturbazioni. La risoluzione del sistema lineare é equivalente alla ricerca dei punti di intersezioni delle rette parametrizzate dalle due equazioni. Poiché le due rette hanno coefficiente angolare prossimo, $-\frac{3}{5} \approx -\frac{3.01}{5.01}$, esse sono quasi parallele. Una piccola perturbazione del coefficiente angolare può allontanare di molto il punto d'intersezione delle due rette dalla posizione originale. In particolare osserviamo che una perturbazione sui dati iniziali dello 0.1% ha causato un errore sulla soluzione del 33%(in norma infinito)

$$\begin{cases} 3x + 5y = 10 \\ 3x + 5.01y = 1 \end{cases}$$

$$A_1 = \begin{bmatrix} 3 & 5 \\ 3.01 & 5.01 \end{bmatrix} \qquad \bar{x}_1 = \begin{bmatrix} -2255 \\ 1355 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} 3 & 5 \\ 3 & 5.01 \end{bmatrix} \qquad \bar{x}_2 = \begin{bmatrix} 1503.3 \\ -900 \end{bmatrix}$$

Studiare il condizionamento del seguente sistema lineare

$$\begin{cases} 5x + 10y = 15 \\ 2x + y = 1 \end{cases}$$

Nel seguente caso le rette formano un angolo di $1\,rad \approx 53°$. Ad una perturbazione dei dati iniziali dello 0.1% corrisponde un errore sulla soluzione dello 0.3%(in norma infinito).

$$\begin{cases} x + 2y = 3 \\ 2x + 1.01y = 1 \end{cases}$$

$$A_1 = \begin{bmatrix} 5 & 10 \\ 2 & 1 \end{bmatrix} \qquad \bar{x}_1 = \begin{bmatrix} -\frac{1}{3} \\ \frac{5}{3} \end{bmatrix}$$

$$A_2 = \begin{bmatrix} 5 & 10 \\ 2 & 1.01 \end{bmatrix} \qquad \bar{x}_2 = \begin{bmatrix} -\frac{103}{299} \\ \frac{500}{299} \end{bmatrix}$$

Parte II
SISTEMI LINEARI

7 ESERCITAZIONE 5.1

7.1 Confronto delle funzioni COND e CONDEST

```
x=[2:100];
for k=2:100
    A = rand(k);
    y1(k-1) = cond(A);
    y2(k-1) = condest(A);
end
figure;
semilogy(x,y1,'r', x,y2, 'b');
legend('cond','condest')
disp('    cond            condest')
disp([y1' y2'])
% Stimare l'errore commesso utilizzando la funzione CONDEST
err = abs(y1-y2)./abs(y1);
figure;
plot(x,err,'r');
disp('    errore CONDEST-COND')
disp(err')
% Stimare il tempo di calcolo
y1=zeros(1,1000);
y2=y1;
for k=1:1000
    A = rand(k);
    tic;
    cond(A,1);
    y1(k) = toc;
    tic;
    condest(A);
    y2(k) = toc;
end
figure;
plot([1:1000],y1,'r', [1:1000],y2, 'b');
legend('tempo cond',' tempo condest')
disp('    tempo cond            tempo condest')
disp([y1' y2'])
```

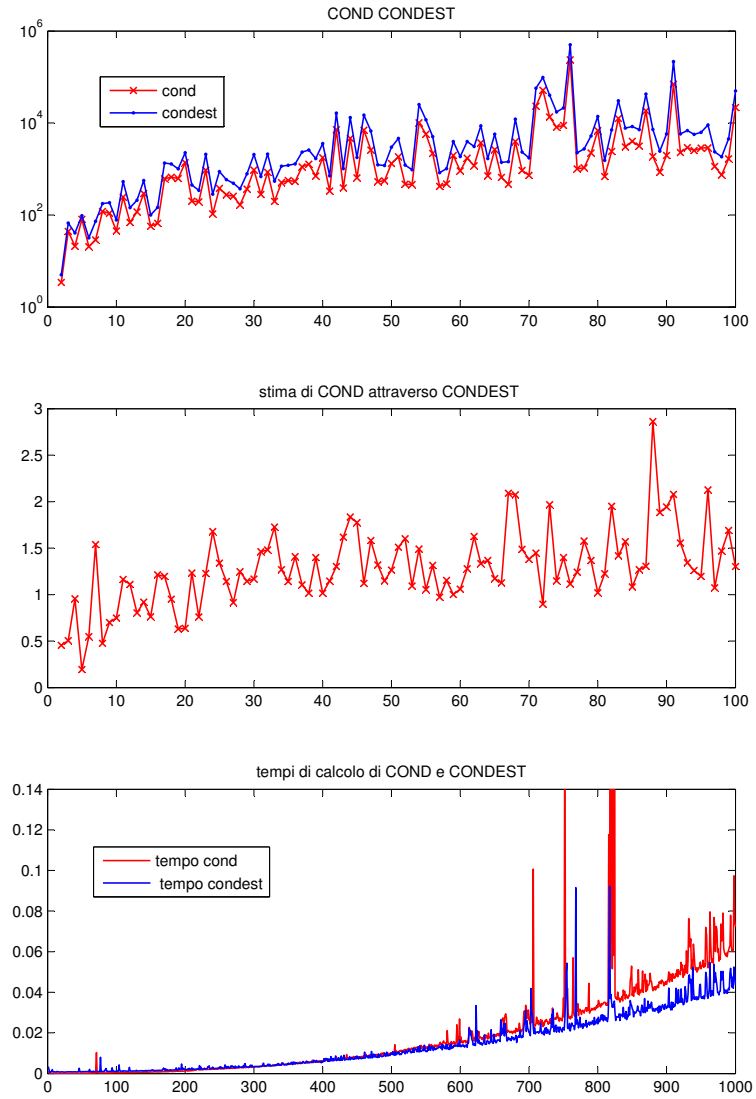

Figura 6: COND CONDEST: errore della stima di COND attraverso CONDEST e confronto dei tempi di calcolo

Osservazione

I grafici precedenti rappresentano rispettivamente la stima data dalla funzione `CONDEST` rispetto alla funzione `COND` per il calcolo del numero di condizionamento di una matrice (in norma 1); l'errore commesso stimando `COND` al posto della funzione `CONDEST`; il tempo di calcolo per le due funzioni per matrici random di dimensione crescente. Si osserva che al crescere della dimensione della matrice l'algoritmo `CONDEST` risulta più efficiente (costo computazionale).

7.2 Condizionamento delle matrici di Hilbert

```
x=[2:29];
for k=2:29
    y1(k-1) = cond(hilb(k),1);
    y2(k-1) = cond(hilb(k),2);
    yinf(k-1) = cond(hilb(k),inf);
end
figure;
semilogy(x,y1,'g', x,y2, 'b', x,yinf, 'r');
legend('K 1','K 2','K inf')
disp('    K 1   K 2  K inf')
disp([y1' y2' yinf'])
```

```
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 2.692153e-017.
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 2.692153e-017.
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 2.739612e-018.
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 2.739612e-018.
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 2.448199e-019.
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 2.448199e-019.
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 1.024999e-018.
```

K 1	K 2	K inf
2.700000000000001e+001	1.928147006790397e+001	2.700000000000001e+001
7.480000000000048e+002	5.240567775860644e+002	7.480000000000048e+002
2.837499999999878e+004	1.551373873892924e+004	2.837499999999879e+004
9.436559999995024e+005	4.766072502433796e+005	9.436559999995024e+005
2.907027900200990e+007	1.495105864148109e+007	2.907027900200990e+007
9.851948900826600e+008	4.753673562966472e+008	9.851948900826600e+008
3.387279198690253e+010	1.525757555001589e+010	3.387279198690253e+010
1.099651961846439e+012	4.931541097528780e+011	1.099651961846439e+012
3.535369067320275e+013	1.602492277132444e+013	3.535369067320276e+013
1.229475923103515e+015	5.225996710750758e+014	1.229475923103516e+015
3.714499490366999e+016	1.677559200768562e+016	3.714499490367000e+016
3.650152020158552e+017	1.759036194677610e+018	3.650152020158552e+017
4.084634689031586e+018	3.082099198249027e+017	4.084634689031586e+018

```
9.756108048351118e+017   4.433272173434449e+017   9.756108048351119e+017
1.028629396772861e+018   3.506508350028904e+017   1.028629396772861e+018
1.038062991158371e+018   1.042961495443520e+018   1.038062991158371e+018
9.288018322422327e+018   6.333486244043811e+017   9.288018322422327e+018
6.280448483776153e+018   5.179790164272418e+018   6.280448483776154e+018
8.654793804464031e+018   2.038253293125565e+018   8.654793804464029e+018
4.848311501936192e+018   5.099251237472185e+018   4.848311501936189e+018
3.160945885991172e+019   1.818107387413698e+018   3.160945885991171e+019
5.378266515975719e+018   1.488357932079489e+019   5.378266515975719e+018
3.759291187310486e+019   5.537324406194551e+018   3.759291187310486e+019
1.696304421535609e+019   6.360142807113324e+018   1.696304421535609e+019
6.744890564933053e+018   1.557368957253072e+019   6.744890564933052e+018
1.197944460734159e+020   6.744000506125791e+018   1.197944460734159e+020
8.253086347347848e+018   3.776680576828867e+018   8.253086347347847e+018
1.469725350593233e+019   2.147565439292347e+018   1.469725350593233e+019
```

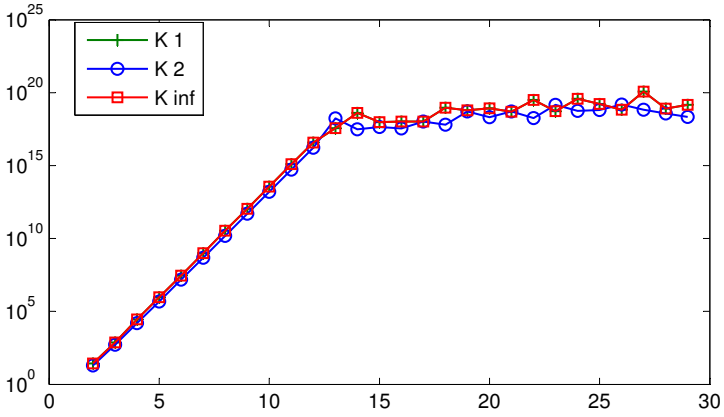

Figura 7: condizionamento delle matrici di Hilbert: `K1 = cond (norma 1); K2 = cond (norma 2); Kinf = cond (norma inf)`

Osservazione

K_1 e K_{inf} coincidono perché norma 1 e norma inf rispettivamente di H e di H^{-1} coincidono. H é una matrice di Hilbert: la somma degli elementi sulla k-esima colonna é pari alla somma degli elementi sulla k-esima riga; giacché la norma 1 e la norma inf sono il massimo delle somme degli elementi rispettivamente di colonna e di riga, devono coincidere. Notiamo come l'ordine di grandezza del condizionamento cresca al crescere della dimensione della matrice

7.3 Condizionamento delle matrici di Vandermonde

```
v1 = [2 3 4];
v2 = [2 2.05 4];
V1 = vander(v1)
V2 = vander(v2)
KV1 = cond(V1,inf);
KV2 = cond(V2,inf);
disp('condizionamento in norma inf di V1 e V2')
disp([KV1 KV2])
```

```
V1 =

     4     2     1
     9     3     1
    16     4     1


V2 =

    4.0000    2.0000    1.0000
    4.2025    2.0500    1.0000
   16.0000    4.0000    1.0000

condizionamento in norma inf di V1 e V2
  1.0e+003 *

    0.3570    3.4672
```

7.4 *esercizio 4*

Costruire un sistema lineare, nota la matrice di partenza A e la sua soluzione x. Calcolare b. Noti A e b, risolvere il sistema lineare associato e confrontare le soluzioni calcolate in aritmetica floating point con quelle esatte. Malgrado ad ogni passo gli errori d'arrotondamento introdotti siano dell'ordine della precisione

di macchina, $\approx 10^{-16}$, essi si propagano ad ogni passo e, nel terzo caso in esame, l'errore relativo é dell'ordine di 10^0: una differenza di 16 ordini di grandezza! Si osserva che l'errore cresce al crescere della dimensione della matrice; fatto ovvio dato che il numero delle operazioni dell'algoritmo é una funzione crescente della dimensione.

```
function result = Hilbert( k )
%HILBERT Analisi a posteriori del sistema lineare Ax=b
%        ove A e' una matrice di Hilbert di dim k
%        e b un vettore di 1 di dim k
    x = ones(k,1);
    A = hilb(k);
    b = A*x;
    result = A\b;
    err = norm(result-x)/norm(result);
    format rat
    disp('Ax = b')
    disp('A')
    disp(A)
    format longe
    disp('x    result    b')
    disp([x'; result'; b']')
    disp('err')
    disp(err)
end
```

```
Ax = b
    k = 7

x       result     b
    1.000000000000000e+000    9.999999999916742e-001    2.592857142857143e+000
    1.000000000000000e+000    1.000000000335732e+000    1.717857142857143e+000
    1.000000000000000e+000    9.999999967411355e-001    1.328968253968254e+000
    1.000000000000000e+000    1.000000012741888e+000    1.095634920634921e+000
    1.000000000000000e+000    9.999999765354278e-001    9.365440115440116e-001
    1.000000000000000e+000    1.000000020349350e+000    8.198773448773450e-001
    1.000000000000000e+000    9.999999932984198e-001    7.301337551337552e-001

err
    1.299825312983505e-008

k = 12

x    result    b
    1.000000000000000e+000    1.000000034504544e+000    3.103210678210678e+000
    1.000000000000000e+000    9.999956469389130e-001    2.180133755133756e+000
    1.000000000000000e+000    1.000136063755542e+000    1.751562326562326e+000
    1.000000000000000e+000    9.981584766937748e-001    1.484895659895660e+000
    1.000000000000000e+000    1.013408909935141e+000    1.297395659895660e+000
    1.000000000000000e+000    9.414726862610096e-001    1.156219189307425e+000
    1.000000000000000e+000    1.162040376601583e+000    1.045108078196314e+000
```

```
    1.000000000000000e+000    7.084489391302908e-001    9.548825142865391e-001
    1.000000000000000e+000    1.339866578894720e+000    8.798825142865392e-001
    1.000000000000000e+000    7.524186105466729e-001    8.163904507944757e-001
    1.000000000000000e+000    1.102420151348244e+000    7.618449962490210e-001
    1.000000000000000e+000    9.816335060207592e-001    7.144141662094954e-001

err
    1.568074062773260e-001

k = 15

x    result    b
    1.000000000000000e+000    9.999999884848885e-001    3.318228993228994e+000
    1.000000000000000e+000    9.999988200010384e-001    2.380728993228994e+000
    1.000000000000000e+000    1.000153035493171e+000    1.939552522640758e+000
    1.000000000000000e+000    9.956783376034795e-001    1.661774744862980e+000
    1.000000000000000e+000    1.055208289152775e+000    1.464406323810348e+000
    1.000000000000000e+000    6.068238310286720e-001    1.314406323810349e+000
    1.000000000000000e+000    2.717791261290424e+000    1.195358704762730e+000
    1.000000000000000e+000   -3.811419278271013e+000    1.097956107360132e+000
    1.000000000000000e+000    9.689857910094865e+000    1.016434368229698e+000
    1.000000000000000e+000   -8.657463514524082e+000    9.469899237852529e-001
    1.000000000000000e+000    6.344376120567810e+000    8.869899237852529e-001
    1.000000000000000e+000    1.692954637371676e+000    8.345423713377005e-001
    1.000000000000000e+000   -2.079670878337648e+000    7.882460750414040e-001
    1.000000000000000e+000    2.820985686699631e+000    7.470372838326128e-001
    1.000000000000000e+000    6.247256512544572e-001    7.100914710247310e-001

err
    9.697842185564345e-001
```

7.5 *esercizio 5*

```
A = [1 1.001;1 1];
b = [2.001;2];
c = [1;0];
x = [2;0];
xesatta = [1;1];
y = [-1001;1000];
yesatta = [-1000;1000];
rx = A*x-b;
ry = A*y-c;
errx = norm(x-xesatta)/norm(xesatta);
erry = norm(y-yesatta)/norm(yesatta);
rxb = norm(rx)/norm(b);
ryc = norm(ry)/norm(c);
K = cond(A);
stimax = rxb*K;
stimay = ryc*K;
disp('A')
disp(A)
```

```
disp('Ax=b')
disp('x xesatta b')
disp([x xesatta b])
disp('Ay=c')
disp('y yesatta c')
disp([y yesatta c])
disp('residuo x e y')
disp([rx ry])
disp('rapporto residuo/termine noto  r(x)/b r(y)/c')
disp([rxb ryc])
disp('condizionamento di A')
disp(K)
disp('stima errore (in eccesso)')
disp([stimax stimay])
disp('errx   erry')
disp([errx erry])
```

```
A
  1.0000e+000  1.0010e+000
  1.0000e+000  1.0000e+000

Ax=b
x xesatta b
  2.0000e+000  1.0000e+000  2.0010e+000
            0  1.0000e+000  2.0000e+000

Ay=c
y yesatta c
       -1001       -1000           1
        1000        1000           0

residuo x e y
 -1.0000e-003 -1.0000e+000
            0 -1.0000e+000

rapporto residuo/termine noto  r(x)/b r(y)/c
  3.5347e-004  1.4142e+000

condizionamento di A
  4.0020e+003

stima errore (in eccesso)
  1.4146e+000  5.6597e+003

errx   erry
  1.0000e+000  7.0711e-004
```

Osservazione

Sia $Ax = b$, A non singolare, $b \neq 0$, $\tilde{x}$ soluzione calcolata in aritmetica floating point, $r(\tilde{x}) = A\tilde{x} - b$ residuo, allora

$$e_{r,x} \leq K(A)\frac{\|r\|}{\|b\|}$$

Il residuo non fornisce una buona stima dell'errore commesso nel calcolo della soluzione del sistema lineare. Benchè il residuo del secondo sistema sia in norma maggiore del primo, l'errore relativo cui è affetto il secondo è di tre ordini di grandezza inferiore a quello del primo. In effetti, nella stima il residuo è pesato con la norma di b. Con tutto ciò non si spiegano ancora i risultati sull'errore ottenuti. Si ricorda, tuttavia, che quella prodotta è una stima in eccesso dell'errore, e in quanto tale può essere pessimistica.

7.6 *esercizio 6*

```
A = [1 1.9999;2 4]
b = [-0.9999 -2]'
x = [1 -1]'
sol = [-101 50]'
r = A*sol-b
err = norm(x-sol)/norm(x)
ratrb = norm(r)/norm(b)
K = cond(A)
```

```
A =

    1.0000    1.9999
    2.0000    4.0000

b = [-0.9999  -2]
x = [1  -1]
sol = [-101  50]
r = [-0.0051  0]

err =   80.6381
r/b =    0.0023
K =  1.2500e+005
```

Osservazione

Sia $Ax = b$, A non singolare, $b \neq 0$, $\tilde{x}$ soluzione calcolata in aritmetica floating point, $r(\tilde{x}) = A\tilde{x} - b$ residuo, allora

$$e_{r,x} \leq K(A)\frac{\|r\|}{\|b\|}$$

Il residuo non fornisce una buona stima dell'errore commesso nel calcolo della soluzione del sistema lineare. Può accadere che ad un piccolo residuo sia associato un errore relativo di qualche ordine di grandezza maggiore. Ciò accade se l'indice di condizionamento é elevato. Si osservi che malgrado ad ogni passo gli errori d'arrotondamento introdotti siano dell'ordine della precisione di macchina, $\approx 10^{-16}$, essi si propagano ad ogni passo e, almeno nel caso in esame, l'errore relativo é dell'ordine di 10^2: una differenza di 18 ordini di grandezza!

8 ESERCITAZIONE 5.2

8.1 Function Gauss

Fattorizzare la matrice A (non-singolare) nel prodotto di una triangolare inferiore L e di una triangolare superiore U

```
function [L,U] = Gauss(A)
%GAUSS Fattorizzazione LU: A=LU ove L e' una matrice triangolare
%      inferiore e U triangolare superiore
    n = max(size(A));
    L = eye(n);
    for k = 1:n-1
        [Mk,Mkinv,flag] = MGauss(A,k,n);
        if(flag)
            L=[]; U=[]; break;
        end
        A = Mk*A;
        L = L*Mkinv;
    end
    U = triu(A);
end

%SUBFUNCTION
function[Mk,Mkinv,flag] = MGauss(A,k,n)
    flag = 0;
    Mk = eye(n);
    if abs(A(k,k))<=eps
        fprintf('elemento pivotale (%d,%d) nullo \n',k,k)
        Mkinv = [];
        flag = 1;
        return
    end
    for i = k+1:n
        Mk(i,k)=-A(i,k)/A(k,k);
    end
    Mkinv = 2*eye(n)-Mk;
end
```

8.2 *esercizio 1*

Valutare la dimensione dello spazio affine delle soluzioni. Esibire la matrice U definita come sopra e la soluzione generata dal comando backslash.

```
A1=[2,6,0,-2;-2,-7,-3,4;-1,-2,0,-1;0,2,10,5]
b1=[-2,-13,7,-10]'
```

```
A2=[12,-15;-4,5]
b2=[8,10]'

A3=[2,3,1,2;1,1,-1,1;1,2,2,1;4,5,-1,4]
b3=[1,2,-1,5]'

Rouche(A1,b1);
Rouche(A2,b2);
Rouche(A3,b3);
```

```
function [U] = Rouche(A,b)
    r = max(size(A))-rank(A);
    if(rank(A)==rank([A,b]))
        if(r==0)
            disp('matrice non-singolare: esiste un unica soluzione')
        else
            fprintf('matrice singolare: il sistema ha soluzioni ...
            inf %d\n',r)
        end
    else
       disp('matrice non-singolare: non esiste soluzione')
    end
    [L,U,P] = lu(A);
    disp('U:')
    disp(U)
    x = A\b;
    disp('soluzione:')
    disp(x)
end
```

```
matrice non-singolare: esiste un unica soluzione
U:
    2.0000    6.0000         0   -2.0000
         0    2.0000   10.0000    5.0000
         0         0   -5.0000   -4.5000
         0         0         0    2.7000

soluzione:
    9.8889
   -5.5556
    3.0000
   -5.7778

matrice non-singolare: non esiste soluzione
U:
    12   -15
```

```
     0     0

Warning: Matrix is singular to working precision.
soluzione:
   Inf
   Inf

matrice singolare: il sistema ha soluzioni inf 2
U:
    4.0000    5.0000   -1.0000    4.0000
         0    0.7500    2.2500         0
         0         0         0         0
         0         0         0         0

Warning: Matrix is singular to working precision.
soluzione:
   NaN
   NaN
   NaN
   NaN
```

Osservazioni

U e A hanno lo stesso rango. Se la matrice A è singolare, lo è anche U e, poiché U è triangolare, esistono elementi diagonali nulli. In particolare, gli zeri in posizioni pivotale occupano le ultime posizioni, data l'implementazione della strategia di risoluzione pivot parziale. Un'eventuale sostituzione backward genererebbe un segnale d'errore (divisione per zero) o restituirebbe `inf` o `NaN` come soluzione del sistema lineare. Se la matrice è singolare, $n - rg(A) = n - rg(U)$ indica la dimensione dello spazio affine delle soluzioni.

8.3 *esercizio 2*

```
A1 = [1,2,3,5;2,-4,6,-7;3,6,-1,0;5,-7,0,2]
Gauss(A1)
A2 = [0,-4,-1;-4,0,3;-1,3,0]
Gauss(A2)
A3 = gallery('lehmer',5)
Gauss(A3)
A4 = gallery('moler',5)
Gauss(A4)
A5 = gallery('minij',5)
Gauss(A5)
B1 = gallery('dorr',5)
Gauss(B1)
```

```
B2 = rand(5)+100*diag(ones(1,5))
Gauss(B2)
B3 = gallery('minij',5)+20*diag(ones(1,5))
Gauss(B3)
```

```
A1 =

     1     2     3     5
     2    -4     6    -7
     3     6    -1     0
     5    -7     0     2

sottomatrice 1:
    -8     0   -17
     0   -10   -15
   -17   -15   -23

sottomatrice 2:
  -10.0000  -15.0000
  -15.0000   13.1250


A2 =

     0    -4    -1
    -4     0     3
    -1     3     0

elemento pivotale (1,1) nullo

A3 ...
A4 ...
A5 =

     1     1     1     1     1
     1     2     2     2     2
     1     2     3     3     3
     1     2     3     4     4
     1     2     3     4     5

sottomatrice 1:
     1     1     1     1
     1     2     2     2
     1     2     3     3
     1     2     3     4
```

```
sottomatrice 2:
     1     1     1
     1     2     2
     1     2     3

sottomatrice 3:
     1     1
     1     2

B1 =

   (1,1)       2.7200
   (2,1)      -0.3600
   (1,2)      -2.3600
   (2,2)       1.7200
   (3,2)      -0.3600
   (2,3)      -1.3600
   (3,3)       0.7200
   (4,3)      -1.3600
   (3,4)      -0.3600
   (4,4)       1.7200
   (5,4)      -2.3600
   (4,5)      -0.3600
   (5,5)       2.7200

sottomatrice 1:
    1.4076   -1.3600         0         0
   -0.3600    0.7200   -0.3600         0
         0   -1.3600    1.7200   -0.3600
         0         0   -2.3600    2.7200

sottomatrice 2:
    0.3722   -0.3600         0
   -1.3600    1.7200   -0.3600
         0   -2.3600    2.7200

sottomatrice 3:
    0.4045   -0.3600
   -2.3600    2.7200

B2 =

  100.9631    0.6241    0.0377    0.2619    0.1068
    0.5468  100.6791    0.8852    0.3354    0.6538
    0.5211    0.3955  100.9133    0.6797    0.4942
```

```
    0.2316    0.3674    0.7962  100.1366    0.7791
    0.4889    0.9880    0.0987    0.7212  100.7150

sottomatrice 1:
  100.6758    0.8850    0.3339    0.6532
    0.3923  100.9131    0.6784    0.4936
    0.3660    0.7961  100.1360    0.7788
    0.9850    0.0985    0.7200  100.7145

sottomatrice 2:
  100.9096    0.6771    0.4911
    0.7929  100.1347    0.7764
    0.0899    0.7167  100.7081

sottomatrice 3:
  100.1294    0.7726
    0.7161  100.7077

B3 =

    21     1     1     1     1
     1    22     2     2     2
     1     2    23     3     3
     1     2     3    24     4
     1     2     3     4    25

sottomatrice 1:
   21.9524    1.9524    1.9524    1.9524
    1.9524   22.9524    2.9524    2.9524
    1.9524    2.9524   23.9524    3.9524
    1.9524    2.9524    3.9524   24.9524

sottomatrice 2:
   22.7787    2.7787    2.7787
    2.7787   23.7787    3.7787
    2.7787    3.7787   24.7787

sottomatrice 3:
   23.4398    3.4398
    3.4398   24.4398
```

Osservazioni

Utilizzando una variante della funzione `Gauss` (sono state inserite opportune stampe a video per le sottomatrici generate nel metodo di Gauss e, nella variante successiva, funzioni per il calcolo e la visualizzazione degli autovalori e della matrice originaria e delle sue sottomatrici), concludiamo che

> *le operazioni del metodo di eliminazione di Gauss preservano nelle sottomatrici da esso generate le proprietà di simmetria e dominanza diagonale della matrice di partenza.*

Le matrici utilizzate sono matrici test dell'ambiente `MATLAB`: `'lehmer'`, `'moler'` e `'minij'` sono matrici simmetriche definite positive; `'dorr'` sono matrici tridiagonali a diagonale dominante.
Se A è definita positiva, allora anche le sottomatrici generate nel metodo di eliminazione di Gauss sono definite positive. A dimostrazione di ciò ne calcoliamo gli autovalori.

```
A3 = gallery('lehmer',5)
Gauss(A3)
A4 = gallery('moler',5)
Gauss(A4)
```

```
A3 =

    1.0000    0.5000    0.3333    0.2500    0.2000
    0.5000    1.0000    0.6667    0.5000    0.4000
    0.3333    0.6667    1.0000    0.7500    0.6000
    0.2500    0.5000    0.7500    1.0000    0.8000
    0.2000    0.4000    0.6000    0.8000    1.0000

autovalori di A3:
    0.1560
    0.2728
    0.5010
    1.0035
    3.0666

sottomatrice 1:
    0.7500    0.5000    0.3750    0.3000
    0.5000    0.8889    0.6667    0.5333
    0.3750    0.6667    0.9375    0.7500
    0.3000    0.5333    0.7500    0.9600

autovalori:
    0.1561
```

```
    0.2773
    0.6077
    2.4953

sottomatrice 2:
    0.5556    0.4167    0.3333
    0.4167    0.7500    0.6000
    0.3333    0.6000    0.8400

autovalori:
    1.6518
    0.3363
    0.1575

sottomatrice 3:
    0.4375    0.3500
    0.3500    0.6400

autovalori:
    0.1744
    0.9031

A4 =

     1    -1    -1    -1    -1
    -1     2     0     0     0
    -1     0     3     1     1
    -1     0     1     4     2
    -1     0     1     2     5

autovalori di A4:
    0.0086
    2.2785
    2.3965
    2.8289
    7.4875

sottomatrice 1:
     1    -1    -1    -1
    -1     2     0     0
    -1     0     3     1
    -1     0     1     4

autovalori:
    0.0334
    2.2974
```

```
    2.5477
    5.1215

sottomatrice 2:
     1    -1    -1
    -1     2     0
    -1     0     3

autovalori:
    0.1206
    2.3473
    3.5321

sottomatrice 3:
     1    -1
    -1     2

autovalori:
    0.3820
    2.6180
```

8.4 *esercizio 3*

Discutere la fattorizzazione di opportune matrici assegnate. Calcolare, a partire dalla fattorizzazione trovata, determinante e rango delle suddette matrici.

```
A1 = [1,2,3,4;2,4,6,8;-1,-2,-3,-1;5,7,0,1];

A2=zeros(5);
v1=2*ones(1,5);
v2=-1*ones(1,4);
A2=A2+diag(v2,-1)+diag(v1)+diag(v2,1);

A3=zeros(5);
v1=0.5*ones(1,5);
v2=-1*ones(1,4);
A3=A3+diag(v2,-1)+diag(v1)+diag(v2,1);

LUP(A1)
LUP(A2)
LUP(A3)

function [] = LUP( A )
    n = max(size(A));
    [L,U,P] = lu(A);
    disp('A:')
```

```
    disp(A)
    disp('L:')
    disp(L)
    disp('U:')
    disp(U)
    disp('P:')
    disp(P)
    rg = 0;
    deter = 1;
    for k = 1:n
        deter = deter*U(k,k);
        if(abs(U(k,k))<=eps)
            tmp = true;
            for i = k:n
                if(abs(U(k,i))>eps)
                    tmp = false;
                end
            end
            if(tmp)
                rg = rg-1;
            end
        end
    end
    fprintf('rg (rg_max:%d): ',n)
    disp(rg)
    disp('det: ')
    disp(deter)
    r = rank(A);
    d = det(A);
    fprintf('Cfr. rank = %d; det = %d\n\n',r,d)
end
```

```
A:
     1     2     3     4
     2     4     6     8
    -1    -2    -3    -1
     5     7     0     1

L:
    1.0000         0         0         0
    0.4000    1.0000         0         0
   -0.2000   -0.5000    1.0000         0
    0.2000    0.5000         0    1.0000
```

```
U:
    5.0000    7.0000         0    1.0000
         0    1.2000    6.0000    7.6000
         0         0         0    3.0000
         0         0         0         0

P:
     0     0     0     1
     0     1     0     0
     0     0     1     0
     1     0     0     0

rg (rg_max:4):      3

det:
     0

Cfr. rank = 3; det = 0

A:
     2    -1     0     0     0
    -1     2    -1     0     0
     0    -1     2    -1     0
     0     0    -1     2    -1
     0     0     0    -1     2

L:
    1.0000         0         0         0         0
   -0.5000    1.0000         0         0         0
         0   -0.6667    1.0000         0         0
         0         0   -0.7500    1.0000         0
         0         0         0   -0.8000    1.0000

U:
    2.0000   -1.0000         0         0         0
         0    1.5000   -1.0000         0         0
         0         0    1.3333   -1.0000         0
         0         0         0    1.2500   -1.0000
         0         0         0         0    1.2000

P:
     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
```

```
rg (rg_max:5):      5

det:
     6

Cfr. rank = 5; det = 6

A:
    0.5000   -1.0000         0         0         0
   -1.0000    0.5000   -1.0000         0         0
         0   -1.0000    0.5000   -1.0000         0
         0         0   -1.0000    0.5000   -1.0000
         0         0         0   -1.0000    0.5000

L:
    1.0000         0         0         0         0
         0    1.0000         0         0         0
         0         0    1.0000         0         0
         0         0         0    1.0000         0
   -0.5000    0.7500    0.8750   -0.3125    1.0000

U:
   -1.0000    0.5000   -1.0000         0         0
         0   -1.0000    0.5000   -1.0000         0
         0         0   -1.0000    0.5000   -1.0000
         0         0         0   -1.0000    0.5000
         0         0         0         0    1.0313

P:
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
     1     0     0     0     0

rg (rg_max:5):      5

det:
    1.0313

Cfr. rank = 5; det = 1.031250e+000
```

Osservazioni

Osserviamo che matrici tridiagonali a diagonale dominante simmetriche si fattorizzano in matrici U e L bidiagonali (e.g. A2). Osserviamo inoltre che non è stata effettuata alcuna permutazione: i metodi di Gauss con o senza pivoting parziale sono equivalenti e parimenti stabili per tale famiglia di matrici. In generale, tuttavia, matrici tridiagonali pur simmetriche non preservano alcuna struttura: A3 ammette U matrice a banda e L somma di una matrice diagonale e di una matrice identicamente nulla salvo l'ultima riga.

8.5 *esercizio 4*

Confrontare la stabilità dei metodi di eliminazione di Gauss (senza scambio, pivot parziale, pivot totale) per il sistema lineare

$$A = \begin{bmatrix} \epsilon & 1 \\ 1 & 1 \end{bmatrix} \qquad b = \begin{bmatrix} 2+\epsilon \\ 4 \end{bmatrix}$$

Si osserva che in generale il metodo di Gauss senza scambio è instabile

```
X1 = zeros(9,2);
X2=X1;
X3=X1;
for k = 2:2:18
    e = 10^(-k);
    A=[e 1; 1 1];
    b=[2+e; 4];
    % Gauss senza scambi
    [L,U]=Gauss(A);
    y=L\b;
    x1=U\y;
    % Gauss pivot parziale
    x2=A\b;
    % Gauss pivot totale
    [L,U,P,Q]=GaussTot(A);
    w=L\(P*b);
    z= U\w;
    x3=Q\z;
    X1(k/2,:)=x1';
    X2(k/2,:)=x2';
    X3(k/2,:)=x3';
end
x=10.^(-[2:2:18])';
disp('Gauss senza scambi: ')
disp('    exp potenza di 10          x(1)              x(2)')
disp([x,X1])
```

```
disp('-----------------------------------------------')
disp('Gauss pivot parziale: ')
disp('    exp potenza di 10          x(1)              x(2)')
disp([x,X2])
disp('-----------------------------------------------')
disp('Gauss pivot totale: ')
disp('    exp potenza di 10          x(1)              x(2)')
disp([x,X3])
```

```
Gauss senza scambi:
    exp potenza di 10                x(1)                x(2)
    1.000000000000000e-002    2.010101010101018e+000    1.989898989898990e+000
    1.000000000000000e-004    2.000100010000061e+000    1.999899989999000e+000
    1.000000000000000e-006    2.000000999924367e+000    1.999998999999000e+000
    1.000000000000000e-008    2.000000010049519e+000    1.999999990000000e+000
    1.000000000000000e-010    2.000000165480742e+000    1.999999999900000e+000
    1.000000000000000e-012    2.000177801164682e+000    1.999999999999000e+000
    1.000000000000000e-014    1.998401444325282e+000    1.999999999999990e+000
    1.000000000000000e-016    4.440892098500626e+000    2.000000000000000e+000
    1.000000000000000e-018                         0    2.000000000000000e+000

-----------------------------------------------
Gauss pivot parziale:
    exp potenza di 10                x(1)                x(2)
    1.000000000000000e-002    2.010101010101010e+000    1.989898989898990e+000
    1.000000000000000e-004    2.000100010001000e+000    1.999899989999000e+000
    1.000000000000000e-006    2.000001000001000e+000    1.999998999999000e+000
    1.000000000000000e-008    2.000000010000000e+000    1.999999990000000e+000
    1.000000000000000e-010    2.000000000100000e+000    1.999999999900000e+000
    1.000000000000000e-012    2.000000000001000e+000    1.999999999999000e+000
    1.000000000000000e-014    2.000000000000010e+000    1.999999999999990e+000
    1.000000000000000e-016    2.000000000000000e+000    2.000000000000000e+000
    1.000000000000000e-018    2.000000000000000e+000    2.000000000000000e+000

-----------------------------------------------
Gauss pivot totale:
    exp potenza di 10                x(1)                x(2)
    1.000000000000000e-002    2.010101010101010e+000    1.989898989898990e+000
    1.000000000000000e-004    2.000100010001000e+000    1.999899989999000e+000
    1.000000000000000e-006    2.000001000001000e+000    1.999998999999000e+000
    1.000000000000000e-008    2.000000010000000e+000    1.999999990000000e+000
    1.000000000000000e-010    2.000000000100000e+000    1.999999999900000e+000
    1.000000000000000e-012    2.000000000001000e+000    1.999999999999000e+000
    1.000000000000000e-014    2.000000000000010e+000    1.999999999999990e+000
    1.000000000000000e-016    2.000000000000000e+000    2.000000000000000e+000
    1.000000000000000e-018    2.000000000000000e+000    2.000000000000000e+000
```

8.6 *esercizio 5*

Confrontare i metodi di eliminazione di Gauss senza scambio, pivot parziale e totale per le seguenti matrici di dimensione crescente

$$a_{i,j} = \begin{cases} 1 & se\ i = j\ o\ se\ j = n \\ -1 & se\ i > j \\ 0 & altrimenti \end{cases}$$

Exempli gratia, per n = 6

```
 1     0     0     0     0     1
-1     1     0     0     0     1
-1    -1     1     0     0     1
-1    -1    -1     1     0     1
-1    -1    -1    -1     1     1
-1    -1    -1    -1    -1     1
```

```
for n = 48:2:58
    A = zeros(n);
    for i = 1:n
        for j=1:n
            if(i==j||j==n)
                A(i,j)=1;
            elseif(i>j)
                A(i,j)=-1;
            end
        end
    end
    x = ones(n,1);
    b = A*x;
    % Gauss senza scambi
    [L,U]=Gauss(A);
    U1=U;
    y=L\b;
    x1=U\y;
    % Gauss pivot parziale
    [L,U2,P2] = lu(A);
    x2 = U2\(L\(P2*b));
    % Gauss pivot totale
    [L,U3,P3,Q]=GaussTot(A);
    w=L\(P3*b);
    z= U3\w;
    x3=Q\z;
```

```
    err1 = norm(x1-x,inf)/norm(x,inf);
    err2 = norm(x2-x,inf)/norm(x,inf);
    err3 = norm(x3-x,inf)/norm(x,inf);
    fprintf('matrice di dim %d: \n',n)
    disp('   err1      err2     err3')
    disp([err1,err2,err3])
    disp('   min U     max U')
    disp([min(min(U1)) max(max(U1)); ...
    min(min(U2)) max(max(U2)); min(min(U3)) max(max(U3))])
    disp('   campo di variabilita dei coefficienti')
    disp(abs([min(min(U1))-max(max(U1)); ...
    min(min(U2))-max(max(U2)); min(min(U3))-max(max(U3))])')
end
figure;
subplot(1,3,1)
spy(U1,15)
title('No pivot')
subplot(1,3,2)
spy(U2,15)
title('pivot parziale')
subplot(1,3,3)
spy(U3,15)
title('pivot totale')
figure;
subplot(1,3,1)
spy(P2,15)
title('P pivot parziale')
subplot(1,3,2)
spy(P3,15)
title('P pivot totale')
subplot(1,3,3)
spy(Q,15)
title('Q pivot totale')
```

```
matrice di dim 48:
   err1      err2     err3
     0     0     0

   min U     max U
            0  1.4074e+014
            0  1.4074e+014
 -2.0000e+000  2.0000e+000

   campo di variabilita dei coefficienti
  1.4074e+014  1.4074e+014  4.0000e+000
```

```
matrice di dim 50:
   err1      err2     err3
     0     0     0

   min U     max U
             0  5.6295e+014
             0  5.6295e+014
 -2.0000e+000  2.0000e+000

   campo di variabilita dei coefficienti
  5.6295e+014  5.6295e+014  4.0000e+000

matrice di dim 52:
   err1      err2     err3
     0     0     0

   min U     max U
             0  2.2518e+015
             0  2.2518e+015
 -2.0000e+000  2.0000e+000

   campo di variabilita dei coefficienti
  2.2518e+015  2.2518e+015  4.0000e+000

Warning: Matrix is close to singular or badly scaled.
RCOND = 1.110223e-016.
matrice di dim 54:
   err1      err2     err3
     0     0     0

   min U     max U
             0  9.0072e+015
             0  9.0072e+015
 -2.0000e+000  2.0000e+000

   campo di variabilita dei coefficienti
  9.0072e+015  9.0072e+015  4.0000e+000

Warning: Matrix is close to singular or badly scaled.
RCOND = 2.775558e-017.
matrice di dim 56:
   err1      err2     err3
     1     1     0

   min U     max U
```

```
           0  3.6029e+016
           0  3.6029e+016
-2.0000e+000  2.0000e+000

   campo di variabilita dei coefficienti
 3.6029e+016  3.6029e+016  4.0000e+000

Warning: Matrix is close to singular or badly scaled.
RCOND = 6.938894e-018.
matrice di dim 58:
   err1      err2     err3
     1     1     0

   min U     max U
           0  1.4412e+017
           0  1.4412e+017
-2.0000e+000  2.0000e+000

   campo di variabilita dei coefficienti
 1.4412e+017  1.4412e+017  4.0000e+000
```

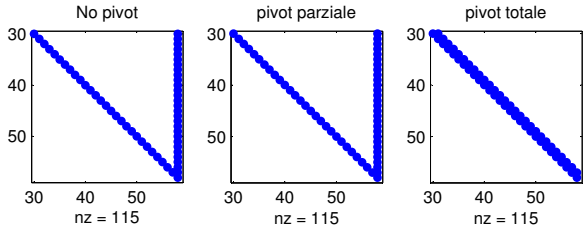

Figura 8

Osservazioni

A partire dal passo n=56, l'errore relativo nel calcolo delle soluzioni attraverso metodo di eliminazione di Gauss senza scambi e con pivot parziale è del 100% (in norma inf; 18-26% in norma 2) per la presenza di errori di arrotondamento [`Matrix badly scaled`]. Gli elementi della matrice U possono variare di 16 ordini di grandezza (cfr. numero delle cifre di mantissa): in sede di sostituzione backward è facile immaginare l'occorrenza di errori di arrotondamento.
Il massimo in norma degli elementi della matrice di partenza è 1; il massimo

in norma degli elementi della matrice U è dell'ordine di 10^{17} (n=58). Pertanto, il metodo di definizione della matrice U senza pivoting totale non riesce a controllare la crescita del modulo degli elementi della matrice U.
L'errore cui è affetta la soluzione ottenuta ricorrendo al metodo di eliminazione di Gauss senza scambi e con pivot parziale è la stessa: la matrice di permutazione P è la matrice identica. Nel caso di pivoting totale P è la matrice identica, ma Q è una matrice quasi-2-diagonale (i.e. la seconda diagonale è piena salvo le prime due entrate). Ciò determina un riarrangiamento della matrice U: nei primi due casi è somma di una matrice diagonale con una matrice identicamente nulla salvo l'ultima colonna; altrimenti è una matrice bidiagonale.
(*Nota* i grafici mostrano la disposizione degli elementi non nulli della matrice corrispondente a n=58; per la costruzione induttiva delle matrici le osservazioni prodotte valgono per ogni n)
L'esempio mostra che il metodo di eliminazione di Gauss con pivoting totale è 'strettamente' più stabile del metodo di eliminazione di Gauss con pivot parziale, i.e. esistono matrici tali per cui l'errore relativo cui sono soggette le soluzioni di un sistema lineare fattorizzato con il primo metodo è strettamente inferiore di quello cui sono affette le soluzioni del sistema fattorizzato con il secondo ($|K(A)| < 26$).

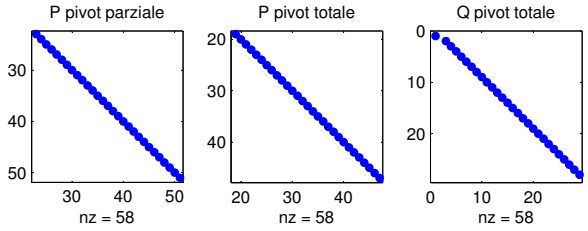

Figura 9

8.7 *esercizio 6*

Confrontare i metodi di eliminazione di Gauss senza scambio, pivot parziale e totale per matrici di Henkel di dimensione crescente

$$a^{(n)}_{i,n+k-i} = \begin{cases} 2^k \quad se \quad k > 0 \\ 2^{1/(2-k)} \quad se \quad k \leq 0 \end{cases}$$

Exempli gratia, per n = 6

```
    1.1225    1.1487    1.1892    1.2599    1.4142    2.0000
    1.1487    1.1892    1.2599    1.4142    2.0000    4.0000
    1.1892    1.2599    1.4142    2.0000    4.0000    8.0000
    1.2599    1.4142    2.0000    4.0000    8.0000   16.0000
    1.4142    2.0000    4.0000    8.0000   16.0000   32.0000
    2.0000    4.0000    8.0000   16.0000   32.0000   64.0000
```

```
err = zeros(3,9);
K = zeros(1,9)';
for n = 4:2:20

    A = zeros(n);
    for i = 1:n
        for k=i+1-n:i
            if(k>0)
                A(i,n+k-i)=2^k;
            elseif(k<=0)
                A(i,n+k-i)=2^(1/(2-k));
            end
        end
    end

    x = ones(n,1);
    b = A*x;
    K((n-4)/2+1) = cond(A);
    % Gauss senza scambi
    [L,U] = Gauss(A);
    U1 = U;
    y = L\b;
    x1 = U\y;
    % Gauss pivot parziale
    [L,U2,P] = lu(A);
    x2 = U2\(L\(P*b));
    % Gauss pivot totale
    [L,U3,P,Q]=GaussTot(A);
    w = L\(P*b);
    z = U3\w;
    x3 = Q\z;

    err(1,k/2) = norm(x1-x,2)/norm(x,2);
    err(2,k/2) = norm(x2-x,2)/norm(x,2);
    err(3,k/2) = norm(x3-x,2)/norm(x,2);
end
disp('condizionamento')
disp(K)
```

```
disp('err1 err2 err3')
disp(err')
x = [1:max(size(err(1,:)))];
figure;
semilogy(x,err(1,:),x,err(2,:),'--',x,err(3,:),'-.')
legend('no pivot','pivot parziale','pivot totale')
```

```
condizionamento
    1.660240054260499e+002
    7.850890907341059e+002
    3.333278901386653e+003
    1.371171212423153e+004
    5.569296306992992e+004
    2.248443637995971e+005
    9.048090053895407e+005
    3.634223654114260e+006
    1.457995907404530e+007

err1 err2 err3
                         0                         0                         0
    2.844100168142183e-015    2.804824172177818e-015    1.888195305854151e-015
    1.829428698092406e-013    1.529511030456415e-014    9.832109866108708e-015
    6.477308792068186e-013    1.000808019232020e-013    1.442510430351958e-014
    4.370661110456514e-012    1.946224572131032e-013    9.398029706290779e-014
    5.792250165329901e-011    3.333046217977088e-012    3.798998189146814e-013
    3.365605860070122e-009    5.940767815771021e-012    1.960678906233370e-012
    8.215915243039073e-008    3.888921349745241e-011    9.621760171782825e-012
    3.737446428872464e-006    1.630921088528963e-010    2.373164186181266e-011
    5.789258371382692e-005    6.616495321223794e-010    4.253446597223911e-011
```

8.8 *esercizio 7*

Costruire una function per il calcolo della soluzione di una generale equazione AX=B, con X, B matrici, che usa la fattorizzazione LU di Matlab.

```
A1=[3 5 7;2 3 4; 5 9 11]
I1=eye(3)
A1inv = GaussMatrix(A1,I1)
disp('Controprova: A*Ainv')
R1 = A1*A1inv
K1 = cond(A1)
errR1 = norm(R1-I1)/norm(I1)
A1inverso = inv(A1)
err1=norm(A1inverso-A1inv)/norm(A1inverso)

A2=zeros(5);
v1=2*ones(1,5);
v2=-1*ones(1,4);
A2=A2+diag(v2,-1)+diag(v1)+diag(v2,1)
```

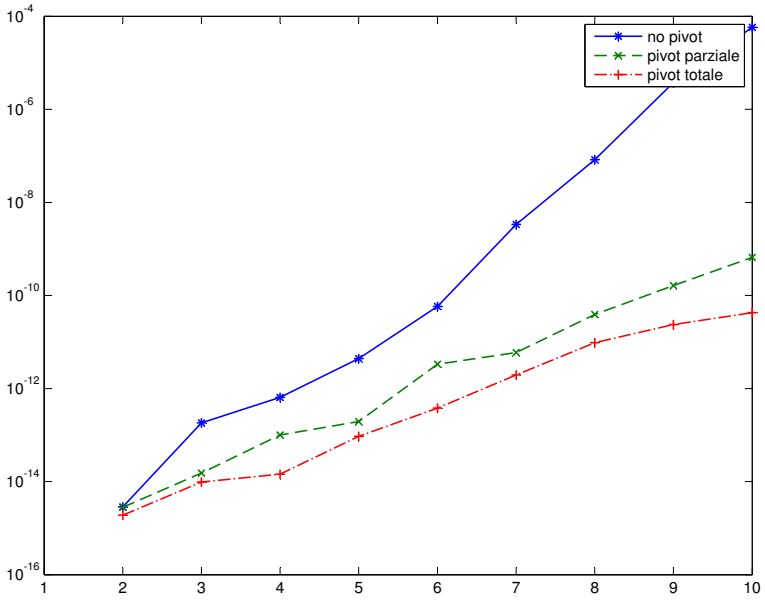

Figura 10: strategie pivotali per matrici di Henkel

```
I2=eye(5)
A2inv = GaussMatrix(A2,I2)
disp('Controprova: A*Ainv')
R2 = A2*A2inv
K2 = cond(A2)
errR2 = norm(R2-I2)/norm(I2)
A2inverso = inv(A2)
err2=norm(A2inverso-A2inv)/norm(A2inverso)
```

```
function [X] = GaussMatrix(A,B)
%GAUSSMATRIX Risolve l'equazione matriciale AX=B
%  % Dati input:
%  A matrice quadrata nxn dei coefficienti del sistema
%  B matrice rettangolare nxm dei termini noti
%
% Dati output:
%  X matrice rettangolare nxm
%.....................................................
% Controlli dimensionali
    if(size(A,2)~=size(B,1))
        fprintf('Errore di dimensione: A matrice %d x %d; ...
        B matrice %d x %d',size(A,1),size(A,2),size(B,1),size(B,2))
        X=[]; return;
    end
    if(size(A,2)~=size(B,2))
        B=B';
        disp('Dimensioni non compatibili: trasporre b')
        scelta = input('Accetti la modifica? Rispondere true ...
        o false: ')
        if(~scelta)
            X=[]; return;
        end
    end
 %.....................................................
    X = zeros(size(B,1),size(B,2));
    for k = 1:size(B,2)
        X(:,k) = A\B(:,k);
    end
end
```

```
A1 =
     3     5     7
     2     3     4
     5     9    11
```

```
I1 =
     1     0     0
     0     1     0
     0     0     1

A1inv =
   -1.5000    4.0000   -0.5000
   -1.0000   -1.0000    1.0000
    1.5000   -1.0000   -0.5000

Controprova: A*Ainv

R1 =
  1.0000e+000  8.8818e-016 -4.4409e-016
            0  1.0000e+000  4.4409e-016
            0  1.7764e-015  1.0000e+000

A1inverso =
   -1.5000    4.0000   -0.5000
   -1.0000   -1.0000    1.0000
    1.5000   -1.0000   -0.5000

K1 =   84.2500
errR1 =  2.3961e-015
err1 =  1.3851e-016



A2 =
     2    -1     0     0     0
    -1     2    -1     0     0
     0    -1     2    -1     0
     0     0    -1     2    -1
     0     0     0    -1     2

I2 =
     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1

A2inv =
    0.8333    0.6667    0.5000    0.3333    0.1667
    0.6667    1.3333    1.0000    0.6667    0.3333
```

```
    0.5000    1.0000    1.5000    1.0000    0.5000
    0.3333    0.6667    1.0000    1.3333    0.6667
    0.1667    0.3333    0.5000    0.6667    0.8333

Controprova: A*Ainv

R2 =
  1.0000e+000 -2.2204e-016 -2.2204e-016 -1.1102e-016 -5.5511e-017
  2.2204e-016  1.0000e+000  6.6613e-016  4.4409e-016  1.1102e-016
            0  2.2204e-016  1.0000e+000            0  1.1102e-016
 -8.3267e-017 -3.3307e-016 -1.1102e-016  1.0000e+000 -2.2204e-016
 -5.5511e-017            0 -2.2204e-016 -2.2204e-016  1.0000e+000

A2inverso =
    0.8333    0.6667    0.5000    0.3333    0.1667
    0.6667    1.3333    1.0000    0.6667    0.3333
    0.5000    1.0000    1.5000    1.0000    0.5000
    0.3333    0.6667    1.0000    1.3333    0.6667
    0.1667    0.3333    0.5000    0.6667    0.8333

K2 =   13.9282
errR2 =  1.1603e-015
err2 =  3.9139e-016
```

8.9 *esercizio 8*

Verificare se le seguenti matrici sono simmetriche definite positive. Osservare che le sottomatrici generate dal metodo di eliminazione di Gauss sono simmetriche definite positive. Osservare che sotto queste ipotesi i metodi di pivoting sono superflui (la matrice di permutazione P è l'identità)

```
A=[10 45 30; 45 20 80; 30 80 171]
FattorizzazioneLU(A);
fprintf('-------------------------------------------------------')
B=[1 -1 -1 -1; -1 2 0 0; -1 0 3 1; -1 0 1 4]
FattorizzazioneLU(B);

function [] = FattorizzazioneLU( A )
    [sA, dA] = defPos(A);
    if(dA)
       [L,U,P] = lu(A);
        disp('A:')
        disp(A)
        disp('L:')
        disp(L)
        disp('U:')
```

```
        disp(U)
        disp('P:')
        disp(P)
        Gauss(A)
    end
end

function [L,U] = Gauss(A)
    for k = 1:n-1
        ...
        fprintf('- SOTTOMATRICE %d\n\n',k)
        Ak = A(k+1:n,k+1:n);
        fprintf('A%d:\n',k)
        disp(Ak)
        defPosShort(Ak);

    end
    ...
end

function [s,d] = defPos( A )
% DEFPOS Valuta se la matrice e' simmetrica definita positiva
%.............................................................
% SIMMETRIA
    fprintf('\nSIMMETRIA\n')
    if(A==A')
        s = true;
        disp('la matrice e' simmetrica');
    else
        s = false;
        disp('la matrice non e' simmetrica')
    end
%.............................................................
% DEFINITA POSITIVA
fprintf('\nDEFINITA POSITIVA\n')
% METODO_1
fprintf('\nMETODO_1: autovalori\n')
    n = max(size(A));
    E = eig(A);
    disp('autovalori:')
    disp(E)
    if(isempty(find(E <= zeros(1,n)',1)))
        disp('la matrice e' simmetrica definita positiva');
    else
        disp('la matrice non e' simmetrica definita positiva')
    end
```

```
% METODO_2
fprintf('\nMETODO_2: minori incapsulati\n')
    D = ones(1,n);
    for k = 1:n
        D(k) = det(A(1:k,1:k));
        fprintf('det minore %d: %d\n',k,D(k))
    end
    if(isempty(find(D <= zeros(1,n) ,1)))
        disp('la matrice e' simmetrica definita positiva');
    else
        disp('la matrice non e' simmetrica definita positiva')
    end
% METODO_3
fprintf('\nMETODO_3: fattorizzazione Cholesky\n')
    [R,p]=chol(A);
    fprintf('flag chol: %d\n',p)
    if(p==0)
        d = true;
        disp('la matrice e' simmetrica definita positiva');
    else
        d = false;
        disp('la matrice non e' simmetrica definita positiva')
    end
fprintf('\n');
end
```

```
A =

    10    45    30
    45    20    80
    30    80   171


SIMMETRIA
la matrice e' simmetrica

DEFINITA POSITIVA

METODO_1: autovalori
autovalori:
   -3.700453713980966e+001
    2.231053060460365e+001
    2.156940065352060e+002

la matrice non e' simmetrica definita positiva
```

```
METODO_2: minori incapsulati
det minore 1: 10
det minore 2: -1825
det minore 3: -178075
la matrice non e' simmetrica definita positiva

METODO_3: fattorizzazione Cholesky
flag chol: 2
la matrice non e' simmetrica definita positiva

------------------------------------------------------------
B =

     1    -1    -1    -1
    -1     2     0     0
    -1     0     3     1
    -1     0     1     4


SIMMETRIA
la matrice e' simmetrica

DEFINITA POSITIVA

METODO_1: autovalori
autovalori:
    3.335894894182129e-002
    2.297411936243863e+000
    2.547709749744224e+000
    5.121519365070091e+000

la matrice e' simmetrica definita positiva

METODO_2: minori incapsulati
det minore 1: 1
det minore 2: 1
det minore 3: 1
det minore 4: 1
la matrice e' simmetrica definita positiva

METODO_3: fattorizzazione Cholesky
flag chol: 0
la matrice e' simmetrica definita positiva
```

```
A:
     1    -1    -1    -1
    -1     2     0     0
    -1     0     3     1
    -1     0     1     4

L:
     1     0     0     0
    -1     1     0     0
    -1    -1     1     0
    -1    -1    -1     1

U:
     1    -1    -1    -1
     0     1    -1    -1
     0     0     1    -1
     0     0     0     1

P:
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1

- SOTTOMATRICE 1

A1:
     1    -1    -1
    -1     2     0
    -1     0     3

SIMMETRIA
la matrice e' simmetrica

DEFINITA POSITIVA
METODO: fattorizzazione Cholesky
flag chol: 0
la matrice e' simmetrica definita positiva

- SOTTOMATRICE 2

A2:
     1    -1
    -1     2
```

```
SIMMETRIA
la matrice e' simmetrica

DEFINITA POSITIVA
METODO: fattorizzazione Cholesky
flag chol: 0
la matrice e' simmetrica definita positiva

- SOTTOMATRICE 3

A3:
     1

SIMMETRIA
la matrice e' simmetrica

DEFINITA POSITIVA
METODO: fattorizzazione Cholesky
flag chol: 0
la matrice e' simmetrica definita positiva
```

Osservazioni

$$U = L^t$$
$$A = LU = U^tU = R^tR \Rightarrow U = R$$

Ad un rapido controllo si mostra che la matrice è costruita affinché gli elementi pivotali di U siano 1.

$$R = diag\left(\frac{1}{\sqrt{u_{i,j}}}\right)U \Rightarrow U = R$$

In generale, se $diag(U) = \begin{bmatrix} 1 & 1 & ... & 1 & 1 \end{bmatrix}$ allora la fattorizzazione LU è equivalente alla fattorizzazione Cholesky. Il metodo di eliminazione di Gauss senza scambi e con pivot parziale sono equivalenti: la matrice di permutazione P è la matrice identica. Infatti, la matrice è simmetrica.

8.10 *esercizio 9*

Verificare se le seguenti matrici sono simmetriche definite positive, nel qual caso fattorizzarle secondo Cholesky.

```
A = [2 1 0; 1 2 1; 0 1 2];
b = [3 4 3]';
[R,p] = chol(A)
```

```
if(p~=0)
    disp('metodo inconsistente: la matrice non e' ...
    simmetrica definita positiva');
else
    disp('la matrice e' simmetrica definita positiva')
    [R,p] = chol(A);
    y = R'\b;
    x = R\y
    disp('Cfr fattorizzazione LU:')
    x = A\b
end
```

```
R =

    1.414213562373095e+000    7.071067811865475e-001                         0
                         0    1.224744871391589e+000    8.164965809277262e-001
                         0                         0    1.154700538379252e+000


p =

     0

la matrice e' simmetrica definita positiva

x =

    9.999999999999996e-001
    1.000000000000000e+000
    9.999999999999998e-001

Cfr fattorizzazione LU:

x =

    9.999999999999996e-001
    1.000000000000000e+000
    9.999999999999998e-001
```

8.11 *esercizio 10*

La fattorizzazione LU determina il riempimento delle matrici sparse, i.e. con un numero elevato di elementi nulli. Tale fenomeno, detto fill-in, determina il calcolo e la memorizzazione di un numero elevato di elementi non nulli.

```
n=100;
e=ones(n,1);
b=[e -e 6*e -e 2*e];
d=[-n/2 -1 0 1 n/2];
A3=spdiags(b,d,n,n);
[L,U,P]=lu(A3)
figure;
subplot(2,3,1)
spy(A3);
subplot(2,3,2)
spy(L);
subplot(2,3,3)
spy(U);
title('U');
A4 = sprand(20,20,.20)
[L,U,P]=lu(A4)
subplot(2,3,4)
spy(A4);
subplot(2,3,5)
spy(L);
subplot(2,3,6)
spy(U);
```

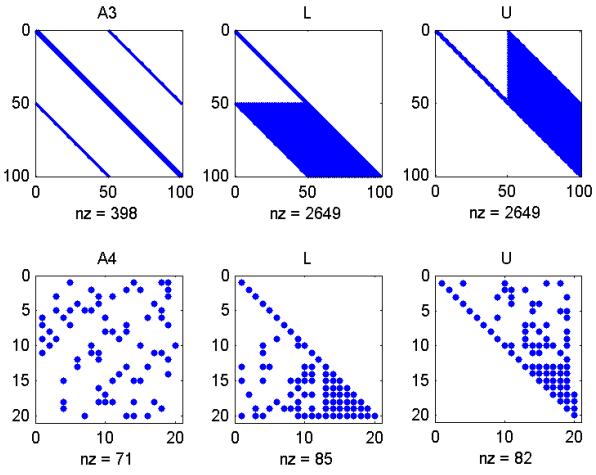

Figura 11

9 ESERCITAZIONE 6

9.1 Algoritmo per la fattorizzazione QR di una matrice

```
function [A,b] = qrsteps(A,b)
% Dati input:
%  A matrice nxm dei coefficienti del sistema
%  b vettore nx1 termine noto
%
% Dati output:
%  R matrice triangolare superiore
%  Q matrice unitaria
%    tali che  Ax=QRx=b
[m,n]=size(A);
if nargin<2, b=zeros(m,1);end
if nargout==0
    clc
    A
    if ~isempty(b)
        b
    end
end
for k = 1:min(m-1,n)
    i = k:m;
    u = A(i,k);
    sigma = norm(u);
    if sigma~=0
        if u(1)~=0
            sigma = sign(u(1))*sigma;
        end
        rho = 1/(sigma*(sigma+u(1)));
        u(1) = u(1)+sigma;
        A(i,k) = 0;
        A(k,k) = -sigma;
        for j=k+1:n
            A(i,j)=A(i,j)-u*(rho*(u'*A(i,j)));
        end
        if ~isempty(b)
            b(i)=b(i)-u*(rho*(u'*b(i)));
        end
    end
     if nargout==0
        A
        if ~isempty(b)
            b
        end
```

Osservazioni

Data una (esiste non unica) fattorizzazione `QR` di una matrice A, le soluzioni del sistema lineare $Ax = b$ (da intendersi secondo le usuali notazioni) possono essere così calcolate:

```
% QRSTEPS: Ax=b
[A,b]=qrsteps(A,b);
x=A\b; % risoluzione backward

% QR: QRx=b
[Q,R]=qr(A);
x=R\(Q'*b);

% QR con pivot per colonne: QR=AE, QREx=b
[Q,R,E]=qr(A);
x=E\(R\(Q'*b));
```

9.2 *esercizio 2*

Osservare la matrice superiore R generata dalla fattorizzazione `QR` di assegnate matrici.
La norma delle matrici R calcolate con le function `qrsteps` o `qr` sono dell'ordine della precisione di macchina. L'opzione `[Q,R,E]=qr(A)` compone opportune permutazioni, riducendo per matrici sparse l'effetto di fill-in e adottando per matrici piene una strategia pivotale per colonne, il cui risultato è l'individuazione di una triangolare superiore R a diagonale ordinata in modulo descrescente. In quest'ultimo caso, la strategia pivotale non stabilizza l'algoritmo, ma piuttosto consente di generare opportune fattorizzazioni, utili ad esempio per il calcolo entro una certa tolleranza del rango di una matrice (a seguire).

```
A1 = [1,2,3,4;2,4,6,8;-1,-2,-3,-1;5,7,0,1];

A2=zeros(5);
v1=2*ones(1,5);
v2=-1*ones(1,4);
A2=A2+diag(v2,-1)+diag(v1)+diag(v2,1);

A3=zeros(5);
v1=0.5*ones(1,5);
v2=-1*ones(1,4);
A3=A3+diag(v2,-1)+diag(v1)+diag(v2,1);

n=100;
e=ones(n,1);
b=[e -e 6*e -e 2*e];
d=[-n/2 -1 0 1 n/2];
```

```
A4=spdiags(b,d,n,n);

A5=sprand(20,20,.20);

format shorte
disp('------------------------------------------------------------')
A1
[R_qrsteps,R_qr,R_qr_pivot]=r(A1)
disp('------------------------------------------------------------')
A2
[R_qrsteps,R_qr,R_qr_pivot]=r(A2)
disp('------------------------------------------------------------')
A3
[R_qrsteps,R_qr,R_qr_pivot]=r(A3)

format longe
[R1,R2,R3]=r(A4);
figure;
subplot(2,2,1)
spy(A4);
title('A4');
subplot(2,2,2)
spy(R1);
title('R qrsteps');
subplot(2,2,3)
spy(R2);
title('R qr');
subplot(2,2,4)
spy(R3);
title('R qr (no fill-in)');

[R1,R2,R3]=r(A5);
figure;
subplot(2,2,1)
spy(A5);
title('A5');
subplot(2,2,2)
spy(R1);
title('R qrsteps');
subplot(2,2,3)
spy(R2);
title('R qr');
subplot(2,2,4)
spy(R3);
title('R qr (no fill-in)');
```

```
function [R1,R2,R3] = r(A)
% Dati input:
%  A matrice quadrata
%
% Dati output:
%  R matrice triangolare superiore
%    tali che QR=A con Q unitaria
%.............................................................
% ATTENZIONE! R non e' unica

[R1,b]=qrsteps(A);
[Q,R2]=qr(A);
[Q,R3,E]=qr(A);

end
```

```
A1 =

     1     2     3     4
     2     4     6     8
    -1    -2    -3    -1
     5     7     0     1

R_qrsteps =

 -5.5678e+000 -8.4414e+000 -3.2329e+000 -4.6697e+000
            0 -1.3198e+000 -6.5991e+000 -7.2590e+000
            0            0 -1.7764e-015 -4.7426e-003
            0            0            0  2.7386e+000

R_qr =

 -5.5678e+000 -8.4414e+000 -3.2329e+000 -4.6697e+000
            0 -1.3198e+000 -6.5991e+000 -7.2590e+000
            0            0 -5.3291e-015 -4.7426e-003
            0            0            0  2.7386e+000

R_qr_pivot =

 -9.0554e+000 -5.4111e+000 -6.9572e+000 -2.8712e+000
            0  6.6121e+000 -2.4899e-001  4.7585e+000
            0            0  2.3528e+000 -3.3611e-001
```

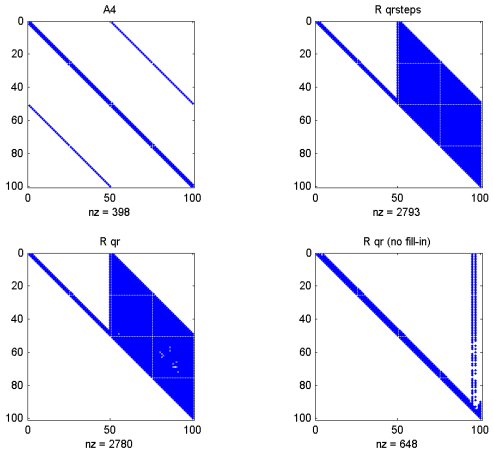

Figura 12

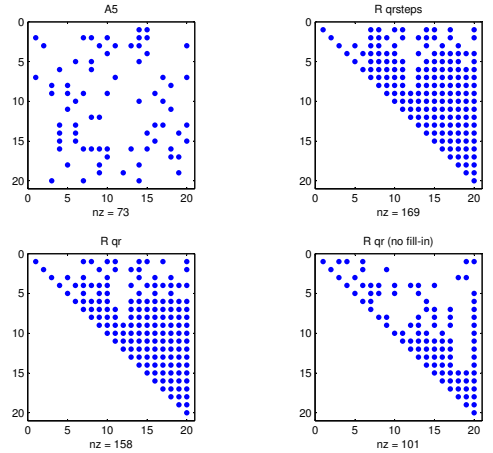

Figura 13

```
           0           0           0  7.6328e-016

------------------------------------------------------------------

A2 =

     2    -1     0     0     0
    -1     2    -1     0     0
     0    -1     2    -1     0
     0     0    -1     2    -1
     0     0     0    -1     2

R_qrsteps =

 -2.2361e+000  1.7889e+000 -4.4721e-001           0           0
           0 -1.6733e+000  1.9124e+000 -5.9761e-001           0
           0           0 -1.4639e+000  1.9518e+000 -6.8313e-001
           0           0           0 -1.3540e+000  1.9695e+000
           0           0           0           0  8.0904e-001

R_qr =

 -2.2361e+000  1.7889e+000 -4.4721e-001           0           0
           0 -1.6733e+000  1.9124e+000 -5.9761e-001           0
           0           0 -1.4639e+000  1.9518e+000 -6.8313e-001
           0           0           0 -1.3540e+000  1.9695e+000
           0           0           0           0  8.0904e-001

R_qr_pivot =

  2.4495e+000  4.0825e-001           0 -1.6330e+000 -1.6330e+000
           0  2.4152e+000 -1.6562e+000  2.7603e-001 -1.3801e+000
           0           0  1.5024e+000  3.0428e-001 -8.5579e-001
           0           0           0 -1.4712e+000  6.9690e-001
           0           0           0           0  4.5883e-001

------------------------------------------------------------------

A3 =

  5.0000e-001 -1.0000e+000           0           0           0
 -1.0000e+000  5.0000e-001 -1.0000e+000           0           0
           0 -1.0000e+000  5.0000e-001 -1.0000e+000           0
           0           0 -1.0000e+000  5.0000e-001 -1.0000e+000
           0           0           0 -1.0000e+000  5.0000e-001
```

```
R_qrsteps =

 -1.1180e+000  8.9443e-001 -8.9443e-001            0            0
            0  1.2042e+000 -1.6609e-001  8.3045e-001            0
            0            0 -1.1926e+000  7.2282e-001 -8.3847e-001
            0            0            0  1.0188e+000 -3.8668e-001
            0            0            0            0  6.3043e-001

R_qr =

 -1.1180e+000  8.9443e-001 -8.9443e-001            0            0
            0  1.2042e+000 -1.6609e-001  8.3045e-001            0
            0            0 -1.1926e+000  7.2282e-001 -8.3847e-001
            0            0            0  1.0188e+000 -3.8668e-001
            0            0            0            0  6.3043e-001

R_qr_pivot =

  1.5000e+000 -6.6667e-001  6.6667e-001 -6.6667e-001            0
            0  1.3437e+000 -4.1345e-001  4.1345e-001  7.4421e-001
            0            0  1.2785e+000  4.8133e-001 -5.4149e-001
            0            0            0 -6.3478e-001  7.4134e-002
            0            0            0            0  6.3043e-001
```

9.3 *esercizio 3*

Confrontare i metodi di eliminazione di Gauss attraverso fattorizzazione `LU` con strategia pivotale parziale e totale e attraverso fattorizzazione `QR` per matrici di Henkel di dimensione crescente

$$a^{(n)}_{i,n+k-i} = \begin{cases} 2^k \quad se \quad k > 0 \\ 2^{1/(2-k)} \quad se \quad k \leq 0 \end{cases}$$

```
err = zeros(3,max(size(4:6:40)));
for n = 4:6:40

    % matrice di Henkel
    A = zeros(n);
    for i = 1:n
        for k=i+1-n:i
            if(k>0)
                A(i,n+k-i)=2^k;
            elseif(k<=0)
                A(i,n+k-i)=2^(1/(2-k));
```

```
            end
        end
    end

    x = ones(n,1);
    b = A*x;
    % Gauss pivot parziale
    [L,U2,P] = lu(A);
    x1 = U2\(L\(P*b));
    % Gauss pivot totale
    [L,U3,P,Q] = GaussTot(A);
    w = L\(P*b);
    z = U3\w;
    x2 = Q\z;
    % Fattorizzazione QR
    [Q,R] = qr(A);
    x3 = R\(Q'*b);

    err(1,(n+2)/6) = norm(x1-x,2)/norm(x,2);
    err(2,(n+2)/6) = norm(x2-x,2)/norm(x,2);
    err(3,(n+2)/6) = norm(x3-x,2)/norm(x,2);
end
disp('Gauss pivot parziale Gauss pivot totale fattorizzazione QR')
disp(err')
x = [1:max(size(err(1,:)))];
figure;
semilogy(x,err(1,:),x,err(2,:),x,err(3,:))
legend('Gauss pivot parziale','Gauss pivot totale','fattorizzazione QR')
```

```
 Gauss pivot parziale      Gauss pivot totale       fattorizzazione QR
2.804824172177818e-015   1.888195305854151e-015   5.212435027073202e-015
1.946224572131032e-013   9.398029706290779e-014   5.415995133254877e-013
3.888921349745241e-011   9.621760171782825e-012   4.486318675197832e-011
2.910268303055751e-009   2.653148284910418e-010   1.474083955074136e-009
2.496662609088788e-007   7.146282710665613e-009   7.060354729941912e-008
2.024969141183910e-005   7.046837091062501e-007   4.427181843649169e-006
1.237155414490810e-003   5.011772373981972e-005   3.144817384617324e-004
```

Osservazioni

Pur privo di strategie pivotali, il metodo di risoluzione dei sistemi lineari attraverso fattorizzazione `QR` è stabile: l'errore relativo sui risultati è compreso (almeno definitivamente) tra quello generato con l'usuale metodo di eliminazione gaussiana con strategia pivotale parziale e totale.

Nota il condizionamento delle matrici cresce all'aumentare della dimensione

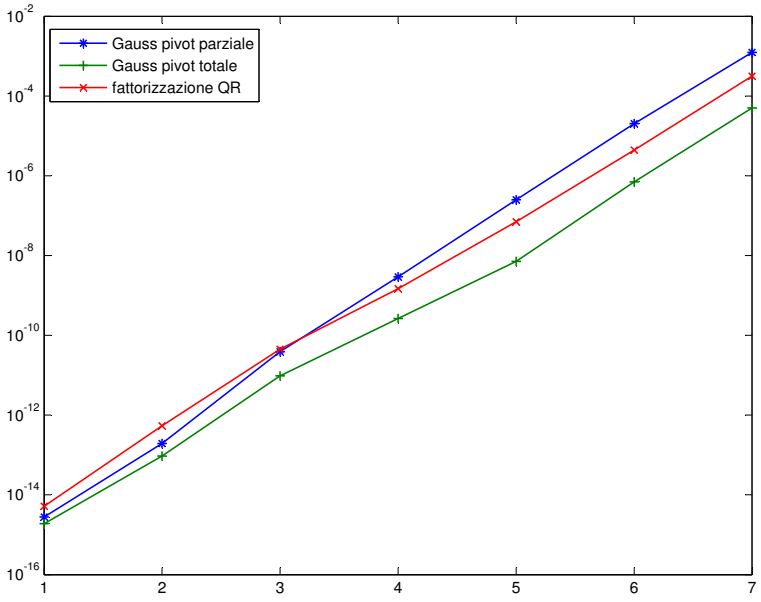

Figura 14: strategie pivotali e fattoriazzaione LU e QR per matrici di Henkel

```
condizionamento:
    1.660240054260499e+002
    1.371171212423153e+004
    9.048090053895407e+005
    5.844767327405380e+007
    3.755611735501062e+009
    2.408582498993631e+011
    1.542996573293626e+013
```

9.4 *esercizio 4*

Confrontare i metodi di eliminazione di Gauss attraverso fattorizzazione `LU` con strategia pivotale parziale e totale e attraverso fattorizzazione `QR` per assegnate matrici.

```
A1 = [1,2,3,4;2,4,6,8;-1,-2,-3,-1;5,7,0,1];

A2=zeros(5);
v1=2*ones(1,5);
v2=-1*ones(1,4);
A2=A2+diag(v2,-1)+diag(v1)+diag(v2,1);

A3=zeros(5);
v1=0.5*ones(1,5);
v2=-1*ones(1,4);
A3=A3+diag(v2,-1)+diag(v1)+diag(v2,1);

% stampa a video
format short
A1
A2
A3

LU_QR(A1);
LU_QR(A2);
LU_QR(A3);


function [] = LU_QR(A)
%LU_QR Stampa a video l'errore relativo nel calcolo delle
%      soluzioni di un sistema lineare con la fattorizzazione
%      LU (pivot parziale e totale) e con la fattorizzazione QR

n = max(size(A));
x = ones(n,1);
b = A*x;
```

```
% Gauss pivot parziale
[L,U2,P] = lu(A);
x1 = U2\(L\(P*b));
% Gauss pivot totale
[L,U3,P,Q] = GaussTot(A);
w = L\(P*b);
z = U3\w;
x2 = Q\z;
% Fattorizzazione QR
[Q,R] = qr(A);
x3 = R\(Q'*b);

format shorte
err(1) = norm(x1-x,2)/norm(x,2);
err(2) = norm(x2-x,2)/norm(x,2);
err(3) = norm(x3-x,2)/norm(x,2);
disp('         err1         err2         err3')
disp(err)
end
```

```
A1 =
     1     2     3     4
     2     4     6     8
    -1    -2    -3    -1
     5     7     0     1

A2 =
     2    -1     0     0     0
    -1     2    -1     0     0
     0    -1     2    -1     0
     0     0    -1     2    -1
     0     0     0    -1     2

A3 =
    0.5000   -1.0000         0         0         0
   -1.0000    0.5000   -1.0000         0         0
         0   -1.0000    0.5000   -1.0000         0
         0         0   -1.0000    0.5000   -1.0000
         0         0         0   -1.0000    0.5000

Warning: Matrix is singular to working precision.
Warning: Matrix is singular to working precision.

         err1         err2         err3
          NaN          NaN  7.2169e-001
```

```
        err1         err2         err3
 4.9651e-017            0  1.5701e-016

        err1         err2         err3
           0            0  2.8522e-016
```

Osservazioni

A1 A1 è singolare. La fattorizzazione `LU` e `QR` conclude; tuttavia la risoluzione backward dei sistemi triangolarizzati produce una divisione per zero (`NaN`). Notiamo per completezza che `err3` è diverso da `NaN`, perché l'elemento diagonale nullo è in verità dell'ordine di 10^{15} per errori di cancellazione. D'altra parte, la scarsa precisione del risultato è segnalata da un errore percentuale del 72%.

A2 errori nell'ordine della precisione di macchina.

A3 errori nell'ordine della precisione di macchina.

9.5 Rango di una matrice

Scrivere una function che assegnata una tolleranza `tol`, stimi il rango di una matrice A sfruttando la fattorizzazione `LU` con strategia pivotale totale e la fattorizzazione `QR` con pivot per colonne.

```
A = [2 -1 0 -2;-1 6 4 12;0 4 5 8;1 0 -1 0]
B = [1 1 1 4 1;-2 -1 0 1 3;-1 0 1 1.7 4;1 1.4 1.8 1 3;0 1 2 3 5]
C = [0.58 -1.1 -0.52; -0.56 1.12 0.56; 0.02 0.02 0.04]
D = magic(4)
E = magic(12)

rango(A)
rango(B)
rango(C)
rango(D)
rango(E)


function []=rango(A)
%RANGO Calcola il rango attraverso fattorizzazione LU e QR
    tol=norm(A)*eps;
    [L,U]=lu(A);
    [Q,R,E]=qr(A);
    k1=0;
    k2=0;
    for i = 1:max(size(A))
```

```
        if abs(U(i,i))<=tol
            k1=k1+1;
        end
        if abs(R(i,i))<=tol
            k2=k2+1;
        end
    end
    k1 = max(size(A))-k1;
    k2 = max(size(A))-k2;
    disp('    LU    QR    rank')
    disp([k1 k2 rank(A)])
end
```

```
A =
     2    -1     0    -2
    -1     6     4    12
     0     4     5     8
     1     0    -1     0

B =
  1.0000e+000  1.0000e+000  1.0000e+000  4.0000e+000  1.0000e+000
 -2.0000e+000 -1.0000e+000            0  1.0000e+000  3.0000e+000
 -1.0000e+000            0  1.0000e+000  1.7000e+000  4.0000e+000
  1.0000e+000  1.4000e+000  1.8000e+000  1.0000e+000  3.0000e+000
            0  1.0000e+000  2.0000e+000  3.0000e+000  5.0000e+000

C =
  5.8000e-001 -1.1000e+000 -5.2000e-001
 -5.6000e-001  1.1200e+000  5.6000e-001
  2.0000e-002  2.0000e-002  4.0000e-002

D =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1

E =
   144     2     3   141   140     6     7   137   136    10    11   133
    13   131   130    16    17   127   126    20    21   123   122    24
    25   119   118    28    29   115   114    32    33   111   110    36
   108    38    39   105   104    42    43   101   100    46    47    97
    96    50    51    93    92    54    55    89    88    58    59    85
    61    83    82    64    65    79    78    68    69    75    74    72
    73    71    70    76    77    67    66    80    81    63    62    84
    60    86    87    57    56    90    91    53    52    94    95    49
    48    98    99    45    44   102   103    41    40   106   107    37
   109    35    34   112   113    31    30   116   117    27    26   120
   121    23    22   124   125    19    18   128   129    15    14   132
    12   134   135     9     8   138   139     5     4   142   143     1
```

```
CALCOLO RANGO: LU QR RANK

    LU    QR    rank
     3     3     3

    LU    QR    rank
     3     3     3

    LU    QR    rank
     2     2     2

    LU    QR    rank
     3     3     3

    LU    QR    rank
     3     3     3
```

10 METODI ITERATIVI

10.1 Metodo di Iacobi e di Gauss-Seidel

```
function [x,k] = iacobi(A,b,tol,Nmax,xo)
% IACOBI Metodo iterativo di Iacobi o degli spostamenti
%        simultanei per la risoluzione del sistema Ax=b
% Dati input:
%  A    matrice quadrata dei coefficienti del sistema
%  b    termine noto del sistema
%  tol  tolleranza
%  Nmax numero massimo di iterazioni
%  xo vettore d'innesco
%
% Dati output:
%  x    soluzione del sistema
%  k    numero di iterazioni compiute

n = size(A,2);
x = zeros(n,1);
err = 1;
k = 0;
while err>tol && k<Nmax
  x(1)=(b(1)-A(1,2:n)*xo(2:n))/A(1,1);
  for i=2:n-1
      x(i)=(b(i)-A(i,1:i-1)*xo(1:i-1)-A(i,i+1:n)*xo(i+1:n))/A(i,i);
  end
  x(n)=(b(n)-A(n,1:n-1)*xo(1:n-1))/A(n,n);
  err = norm(x-xo,inf);
  k = k+1;
  xo = x;
end
if k==Nmax
    fprintf('Metodo non converge in %d iterazioni\n',Nmax)
end
end
```

```
function [x,k] = gseidel(A,b,tol,Nmax,xo)
% GSEIDEL Metodo iterativo di Gauss-Seidel
%         per la risoluzione del sistema Ax=b
% Dati input:
%  A    matrice quadrata dei coefficienti del sistema
%  b    termine noto del sistema
%  tol  tolleranza
%  Nmax numero massimo di iterazioni
```

```
%  xo vettore d'innesco
%
% Dati output:
%  x    soluzione del sistema
%  k    numero di iterazioni compiute

n = size(A,2);
x = xo;
err = 1;
k = 0;
while err>tol && k<Nmax
  x(1)=(b(1)-A(1,2:n)*x(2:n))/A(1,1);
  for i=2:n-1
    x(i)=(b(i)-A(i,1:i-1)*x(1:i-1)-A(i,i+1:n)*x(i+1:n))/A(i,i);
  end
  x(n)=(b(n)-A(n,1:n-1)*x(1:n-1))/A(n,n);
  err = norm(x-xo,inf);
  k = k+1;
  xo = x;
end
if k==Nmax
    fprintf('Metodo non converge in %d iterazioni\n',Nmax)
end
end
```

10.2 *esercizio 2*

Confrontare i metodi di Iacobi e Gauss-Seidel per assegnate matrici. Le function precedenti sono integrate dal calcolo del raggio spettrale e della norma delle matrici di iterazione, rispettivamente:

```
    P = diag(1./diag(A))*(A-diag(diag(A)));
    r = max(abs(eig(P)));
    nP = norm(P,inf);

    P = inv(tril(A))*(A-tril(A));
    r = max(abs(eig(P)));
    nP = norm(P,inf);
```

```
tol=10^(-6);
Nmax=1000;
xo=zeros(1,3)';
A=[20 2 -1; 2 13 -2; 1 1 1];
b=[25 30 2]';
metodi_iterativi_stampa(A,b,tol,Nmax,xo);

A=[2 13 -2; 20 2 -1; 1 1 1];
```

```
b=[30 25 2]';
metodi_iterativi_stampa(A,b,tol,Nmax,xo);

function [] = metodi_iterativi_stampa(A,b,tol,Nmax,xo)

[x1,k1,r1,nP1] = iacobi(A,b,tol,Nmax,xo);
[x2,k2,r2,nP2] = gseidel(A,b,tol,Nmax,xo);

disp('SISTEMA: Ax=b')
A
b=b'
fprintf('Soluzioni: GAUSS PIVOT PARZIALE')
x=(A\b')'
fprintf('Soluzioni: IACOBI [#iterazioni: %d]',k1)
x1
fprintf('raggio spettrale della matrice di iterazione: %d\n',r1)
fprintf('norma della matrice di iterazione: %d\n',nP1)
fprintf('Soluzioni: GAUSS-SEIDEL [#iterazioni: %d]',k2)
x2
fprintf('raggio spettrale della matrice di iterazione: %d\n',r2)
fprintf('norma della matrice di iterazione: %d\n',nP2)
fprintf('tolleranza: %d; #iterazioni massime: %d\n',tol,Nmax)
disp('------------------------------------------------------------')
end
```

```
SISTEMA: Ax=b

A =
    20     2    -1
     2    13    -2
     1     1     1
b =
    25    30     2

Soluzioni: GAUSS PIVOT PARZIALE
x =
     1     2    -1

Soluzioni: IACOBI [#iterazioni: 21]
x1 =
    9.999999933848252e-001
    2.000000033968707e+000
   -1.000000330399930e+000

raggio spettrale della matrice di iterazione: 4.489641e-001
norma della matrice di iterazione: 2
```

```
Soluzioni: GAUSS-SEIDEL [#iterazioni: 12]
x2 =
    9.999999730715231e-001
    1.999999965198033e+000
   -9.999999382695557e-001

raggio spettrale della matrice di iterazione: 2.438577e-001
norma della matrice di iterazione: 2.807692e-001
tolleranza: 1.000000e-006; #iterazioni massime: 1000
-----------------------------------------------------------------
SISTEMA: Ax=b

A =
     2    13    -2
    20     2    -1
     1     1     1
b =
    30    25     2

Soluzioni: GAUSS PIVOT PARZIALE
x =
     1     2    -1

Soluzioni: IACOBI [#iterazioni: 341]
x1 =
   Inf
  -Inf
   Inf

raggio spettrale della matrice di iterazione: 8.071039e+000
norma della matrice di iterazione: 1.050000e+001
Soluzioni: GAUSS-SEIDEL [#iterazioni: 166]
x2 =
   Inf
   NaN
   NaN

raggio spettrale della matrice di iterazione: 7.354419e+001
norma della matrice di iterazione: 7.450000e+001
tolleranza: 1.000000e-006; #iterazioni massime: 1000
-----------------------------------------------------------------
```

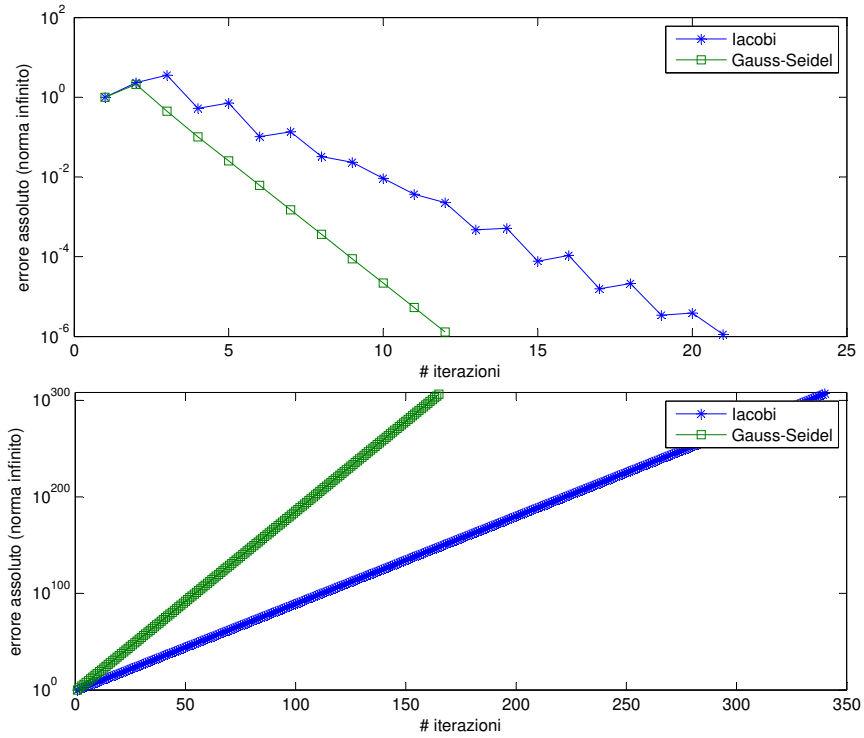

Figura 15: convergenza dei metodi di Iacobi e di Gauss-Seidel - norma infinito dell'incremento come test d'arresto

Osservazioni

1. Entrambi i metodi convergono, ma GS converge più velocemente di J: $\rho(P_{GS}) < \rho(P_J) < 1$. Al contrario di I, GS converge in modo monotono rispetto alla norma infinito ($e_{abs,\infty}(n)$ *monotona*, i.e. l'errore assoluto è funzione monotona del numero di iterazioni): $\|P_J\| > 1$, $\|P_{GS}\| < 1$.

2. Entrambi i metodi non convergono: $\rho(P_{GS})$, $\rho(P_J) > 1$. Escono dal ciclo prima di aver raggiunto il numero massimo di iterazioni a causa del raggiungimento della soglia di overflow. GS diverge più velocemente: $\rho(P_{GS}) > \rho(P_J)$

10.3 *esercizio 3-4*

Scrivere una function dal nome `jacobsparse.m` che implementi il metodo di Jacobi per la risoluzione di un sistema lineare $Ax = b$ con A memorizzata nel formato sparse di `MATLAB` . Confrontare il tempo impiegato dalle due implementazioni del metodo di Jacobi (function `jacobsparse.m` e function `jacobi.m`) e dal metodo di eliminazione gaussiana realizzato dal comando \ di `MATLAB`, quando applicati a un sistema lineare con matrice sparsa memorizzata rispettivamente nei formati sparso e pieno.

```
tol=10^(-6);
Nmax=2000;
i=1;
t = zeros(6,4);
for n = [1000 2000 4000 5000 7000 8000]
    [S,F] = sparsa(n);

    xo = zeros(n,1);
    b = ones(n,1);

    [x,k,t1] = iacobi(F,b,tol,Nmax,xo);
    t(i,1)= t1;

    [x,k,t2] = jacobsparse(S,b,tol,Nmax,xo);
    t(i,2)= t2;

    tic;
    x=F\b;
    t(i,3)=toc;

    tic;
    x=S\sparse(b);
    t(i,4)=toc;

    i = i+1;
```

```
end
disp('tempo di calcolo:')
disp('Iac full    Iac sparse    Gauss full   Gauss sparse')
disp(t)

function [x,k,t] = jacobsparse(A,b,tol,Nmax,xo)
% IACOBSPARSE Metodo iterativo di Iacobi per il
%             calcolo della soluzione del sistema Ax=b
%             ove A  una matrice sparsa
% Dati input:
%  A    matrice quadrata dei coefficienti del sistema
%  b    termine noto del sistema
%  tol  tolleranza
%  Nmax numero massimo di iterazioni
%  xo vettore d'innesco
%
% Dati output:
%  x    soluzione del sistema
%  k    numero di iterazioni compiute
%  t    tempo di calcolo (costo computazionale)

A = sparse(A); % ctrl A sparsa
b = sparse(b);
xo = sparse(xo);

D = diag(diag(A));
Dinv = sparse(diag(1./diag(A)));
B = -Dinv*(A-D);
C = Dinv*b;

n = size(A,2);
x = zeros(n,1);
err = 1;
k = 0;
tic;
while err>tol && k<Nmax
    x = B*xo+C;
    err = norm(x-xo);
    k = k+1;
    xo = x;
end
t = toc;
if k==Nmax
    fprintf('Metodo non converge in %d iterazioni\n',Nmax)
end
end
```

```
function [S,F] = sparsa(n)
%SPARSA genera una matrice sparsa di dim n
    e = ones(n,1);
    b = [e -e 6*e -e 2*e];
    d = [-n/2 -1 0 1 n/2];
    S = spdiags(b,d,n,n)./100;
    F = full(S);
end
```

```
function [S,F] = sparsa(n)
%SPARSA genera una matrice sparsa di dim n
    S = sparse(sparse(-10^(-4)*abs(sprand(n,n,.000001)))+eye(n));
    F = full(S);
end
```

```
function [S,F] = sparsa(n)
%SPARSA genera una matrice sparsa di dim n
    S = sparse(sparse(-10^(-4)*abs(sprand(n,n,.20)))+eye(n));
    F = full(S);
end
```

```
SPDIAGS
tempo di calcolo:
  Iac full     Iac sparse   Gauss full   Gauss sparse
  5.9741e-001  1.4756e-003  5.4333e-002  1.8679e-003
  2.1687e+000  3.0508e-003  3.1737e-001  3.2574e-003
  1.0011e+001  6.1662e-003  1.3836e+000  6.4462e-003
  1.5374e+001  7.9519e-003  2.2248e+000  8.1562e-003
  3.0572e+001  1.1210e-002  4.8809e+000  1.1483e-002
  4.5846e+001  1.2928e-002  6.5794e+000  1.2711e-002

SPRAND (density: .000001)
tempo di calcolo:
  Iac full    Iac sparse    Gauss full   Gauss sparse
  5.1984e-002  9.1705e-005  1.4285e-003  3.2577e-004
  1.9746e-001  1.0760e-004  1.7533e-002  2.2214e-004
  8.8668e-001  1.5825e-004  6.7953e-002  1.2643e-003
  1.3609e+000  1.8506e-004  1.0689e-001  1.3676e-003
  2.6196e+000  2.3241e-004  2.0466e-001  1.6388e-003
  3.9230e+000  2.6022e-004  2.7347e-001  1.6924e-003
```

```
SPRAND (density: .20)
tempo di calcolo:
  Iac full    Iac sparse    Gauss full   Gauss sparse
  7.5785e-002  1.2978e-003  2.4341e-002  9.1539e-002
  3.3158e-001  7.2437e-003  1.3930e-001  4.8154e-001
  1.7458e+000  3.8931e-002  8.0987e-001  2.4019e+000
  2.6791e+000  5.8841e-002  1.4506e+000  4.3137e+000
  5.8967e+000  1.3891e-001  3.5917e+000  9.8287e+000
  8.8815e+000  1.8302e-001  5.1462e+000  1.3102e+001
```

Osservazioni

In generale, il tempo di calcolo delle soluzioni di un sistema lineare di una matrice memorizzata in formato sparso è inferiore al tempo di calcolo delle soluzioni dello stesso sistema memorizzato in formato pieno. Sono state scelte due classi di matrici sparse:

1. Matrici sparse pentadiagonali: i metodi Gauss sparse e Iacobi sparse hanno simile costo computazionale. Il metodo di Gauss ottimizzato cerca di ridurre l'effetto di fill-in componendo opportune permutazioni (cfr. `[Q,R,E]=qr(A)`)

2. Matrici sparse random (a diagonale dominante): i metodi iterativi applicati a matrici salvate in formato sparso hanno costi computazionali inferiori

   densità .000001**.** La densità è talmente piccola che l'effetto di fill-in è molto limitato; il metodo di Iacobi sparse ha il costo computazionale inferiore.

   densità .20**.** L'effetto di fill-in è esteso: la matrice U del metodo di eliminazione di Gauss è vicina ad una matrice triangolare superiore completamente densa; il metodo di Iacobi sparse ha il costo computazionale inferiore.
   Il peggioramento del costo computazionale di Gauss sparse è dovuto alla gestione di matrici dense di grande dimensione in formato sparso.

	nnz**A**	#**A**	nnz**U**	#**U**
1000	182219	1000000	491922	501000
2000	726885	4000000	1986538	2002000
4000	2903585	16000000	7974166	8004000
5000	4535609	25000000	12457121	12505000
7000	8887987	49000000	24427637	24507000
8000	11607947	64000000	31931328	32008000

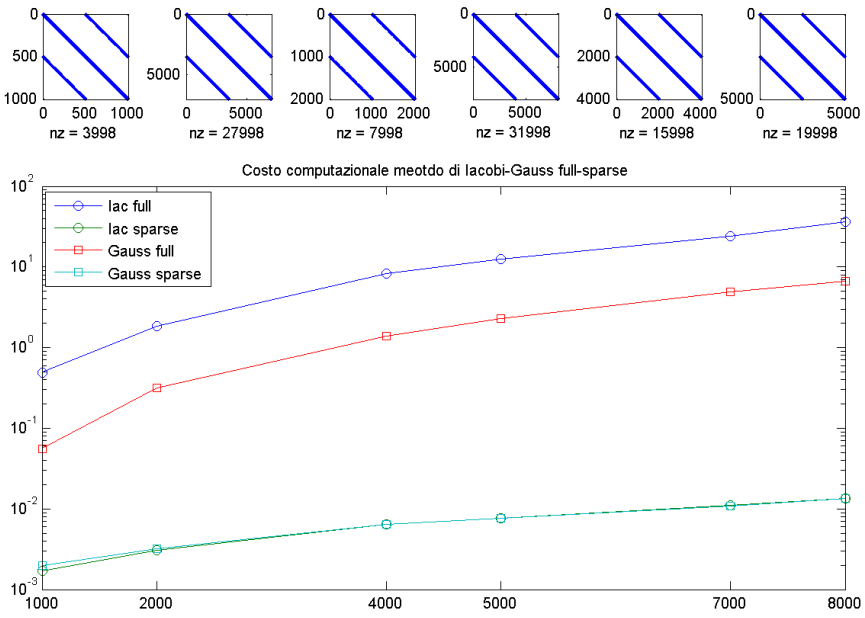

Figura 16: costo computazionale del metodo di Iacobi e di Gauss-Seidel applicati a matrici pentadiagonali

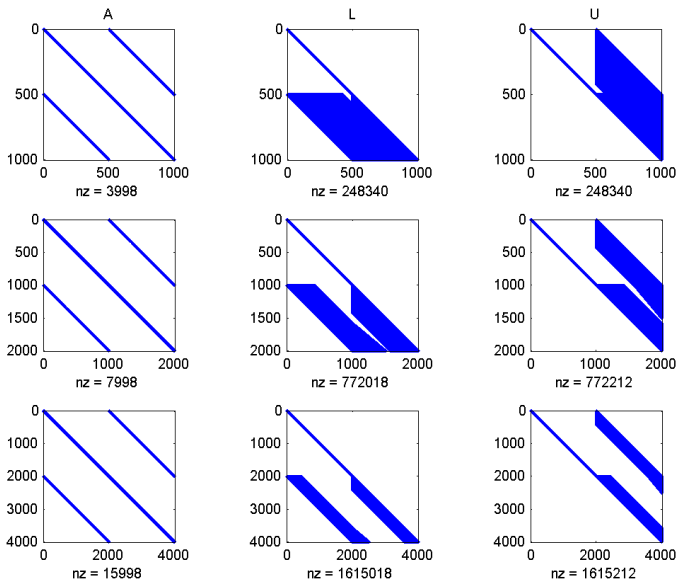

Figura 17: effetto fill-in relativo alla fattorizzazione LU di matrici pentadiagonali di dimensione crescente 1000, 2000, 4000

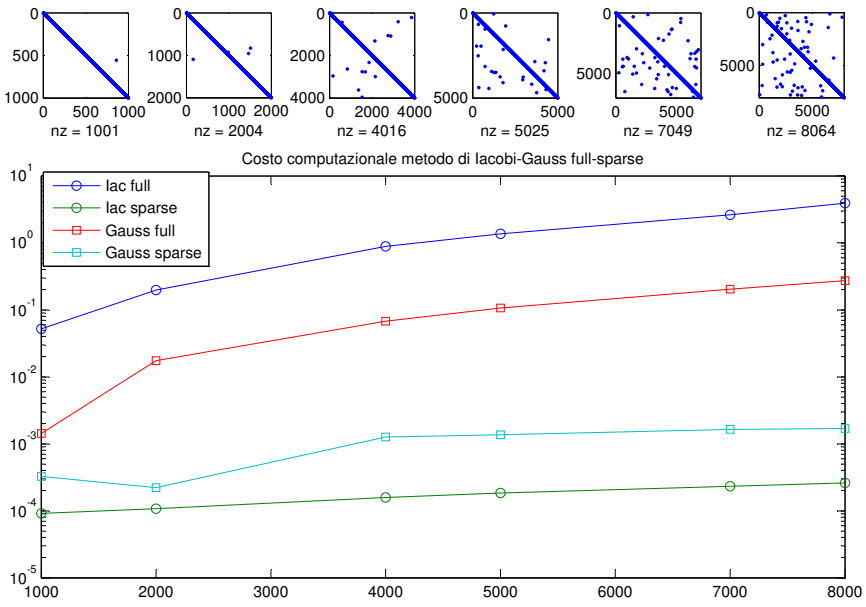

Figura 18: costo computazionale del metodo di Iacobi e di Gauss-Seidel applicati a matrici sparse random (densità .000001)

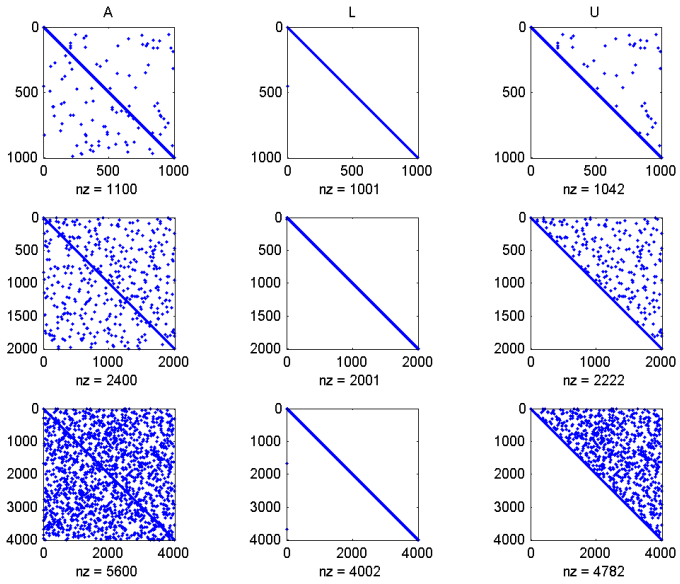

Figura 19: effetto fill-in relativo alla fattorizzazione LU di matrici sparse random di dimensione crescente 1000, 2000, 4000 [matrix A,P,U]

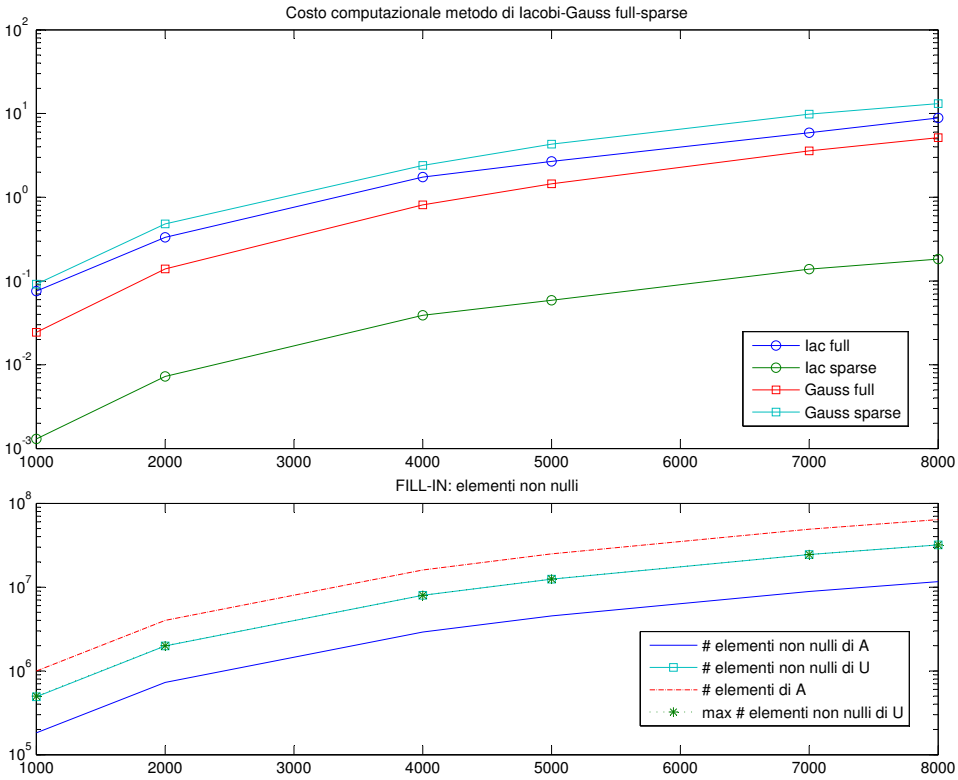

Figura 20: costo computazionale del metodo di Iacobi e di Gauss-Seidel applicati a matrici sparse random (densità .20)

10.4 *esercizio 5*

Le function utilizzate sono le stesse utilizzate nell'esercizio 2, integrate dal calcolo del raggio spettrale delle matrici di iterazione rispettivamente:

```
r = max(abs(eig(inv(tril(A))*(A-tril(A)))));
r = max(abs(eig(diag(1./diag(A))*(A-diag(diag(A))))));
```

```
tol=10^(-6);
Nmax=100;
xo=zeros(1,3)';

A=[2 -1 1; 2 2 2; -1 -1 2];
b=[2 1 3]';
metodi_iterativi_stampa(A,b,tol,Nmax,xo);

A=[1 -2 2; -1 1 -1; -2 -2 1];
b=[1 -1 3]';
metodi_iterativi_stampa(A,b,tol,Nmax,xo);

A=[4 1 1; 2 -9 0; 0 -8 -6];
b=[6 -7 14]';
metodi_iterativi_stampa(A,b,tol,Nmax,xo);

A=[7 6 9; 4 5 -4; -7 -3 8];
b=[22 5 -2]';
metodi_iterativi_stampa(A,b,tol,Nmax,xo);
```

```
SISTEMA: Ax=b

A =
     2    -1     1
     2     2     2
    -1    -1     2
b =
     2     1     3

Soluzioni: GAUSS PIVOT PARZIALE
x =
    5.555555555555547e-002
    -7.222222222222223e-001
    1.166666666666667e+000

Soluzioni: IACOBI [#iterazioni: 100]
x1 =
   -2.491191714355230e+004
```

```
    7.162119928771288e+004
   -6.072176678740874e+004

raggio spettrale della matrice di iterazione: 1.118034e+000
norma della matrice di iterazione: 2
Soluzioni: GAUSS-SEIDEL [#iterazioni: 27]
x2 =
    5.555534362792969e-002
   -7.222219929099083e-001
    1.166666675359011e+000

raggio spettrale della matrice di iterazione: 5.000000e-001
norma della matrice di iterazione: 1
tolleranza: 1.000000e-006; #iterazioni massime: 100
------------------------------------------------------------------
SISTEMA: Ax=b

A =
     1    -2     2
    -1     1    -1
    -2    -2     1
b =
     1    -1     3

Soluzioni: GAUSS PIVOT PARZIALE
x =
    1.000000000000002e+000
    -5.000000000000004e+000
    -5.000000000000005e+000

Soluzioni: IACOBI [#iterazioni: 4]
x1 =
     1
    -5
    -5

raggio spettrale della matrice di iterazione: 1.080934e-005
norma della matrice di iterazione: 4
Soluzioni: GAUSS-SEIDEL [#iterazioni: 100]
x2 =
   -8.794019615952118e+067
   -3.642602192702404e+067
   -2.487324361730905e+068

raggio spettrale della matrice di iterazione: 4.828427e+000
norma della matrice di iterazione: 14
```

```
tolleranza: 1.000000e-006; #iterazioni massime: 100
------------------------------------------------------------------
SISTEMA: Ax=b

A =
     4     1     1
     2    -9     0
     0    -8    -6
b =
     6    -7    14

Soluzioni: GAUSS PIVOT PARZIALE
x =
    2.188679245283019e+000
    1.264150943396226e+000
    -4.018867924528301e+000

Soluzioni: IACOBI [#iterazioni: 17]
x1 =
    2.188679301027777e+000
    1.264150874621463e+000
   -4.018867382279180e+000

raggio spettrale della matrice di iterazione: 4.438188e-001
norma della matrice di iterazione: 1.333333e+000
Soluzioni: GAUSS-SEIDEL [#iterazioni: 6]
x2 =
    2.188679243783168e+000
    1.264150943062926e+000
   -4.018867924083902e+000

raggio spettrale della matrice di iterazione: 1.851852e-002
norma della matrice di iterazione: 5.000000e-001
tolleranza: 1.000000e-006; #iterazioni massime: 100
------------------------------------------------------------------
SISTEMA: Ax=b

A =
     7     6     9
     4     5    -4
    -7    -3     8
b =
    22     5    -2

Soluzioni: GAUSS PIVOT PARZIALE
x =
```

```
     1     1     1

Soluzioni: IACOBI [#iterazioni: 35]
x1 =
    1.000000529161942e+000
    9.999993148164302e-001
    1.000000599050830e+000

raggio spettrale della matrice di iterazione: 6.411328e-001
norma della matrice di iterazione: 2.142857e+000
Soluzioni: GAUSS-SEIDEL [#iterazioni: 57]
x2 =
    9.999985758377515e-001
    1.000001262911208e+000
    9.999992274497356e-001

raggio spettrale della matrice di iterazione: 7.745967e-001
norma della matrice di iterazione: 2.514286e+000
tolleranza: 1.000000e-006; #iterazioni massime: 100
-----------------------------------------------------------------
```

Osservazioni

A seguire rappresentiamo la stima dell'errore assoluto (test d'arresto dell'incremento), i.e. la norma infinito della distanza tra due iterate successive, in funzione del numero d'iterazioni per il metodo di Iacobi e di Gauss-Seidel. I due metodi non hanno in generale lo stesso carattere (convergenza) e, se convergono, non è possibile confrontarne a priori le velocità di convergenza.
Il tasso asintotico di convergenza è misura di quanto velocemente si guadagnino cifre significative: $R = -\log_{10}(\rho(P))$. Il raggio spettrale della matrice di iterazione ($\rho(P)$) è dunque misura della velocità (asintotica) di convergenza.
La monotonia delle funzioni dipende dalla norma della matrice di iterazione: *se* $\|P\|_\infty < 1 \Rightarrow e_{ass\,(\infty)}(n)$ *monotona*. Se la norma della matrice è maggiore di 1, l'errore assoluto all'iterata successiva può crescere poiché $\|e^{(k)}\| \leq \|P\|\|e^{(k+1)}\|$; tuttavia, se il metodo converge ($\Leftrightarrow \rho(P) < 1$), malgrado le oscillazioni, $\lim_{n\to+\infty} e_{ass\,(\infty)}(n) = 0$. A seguire confrontiamo i risultati ottenuti con il test d'arresto dell'incremento in norma 2 e con il test d'arresto del residuo. Il controllo è reso necessario dal fatto che i risultati del primo sistema non sono a priori affidabili: $\|P\|_\infty = 1$ e la stima dell'errore assoluto attraverso la norma della distanza tra le due iterate successive non è significativa.

$$\|e_{ass}^{(k)}\| \leq \frac{\|P\|}{1-\|P\|}\|x^{(k)} - x^{(k-1)}\|$$

$$\|e_{rel}^{(k)}\| \leq K(A)\frac{\|r^{(k)}\|}{\|b\|}$$

Cambiando norma è possibile rimuovere l'ostacolo. Il test d'arresto del residuo mostra che nella specificità del caso in esame il fatto che $\|P\|_\infty = 1$ non compromette la determinazione del numero utile di iterazioni per il calcolo della soluzione. D'altra parte, il buon condizionamento del problema (cfr. numero di condizionamento di A) consente di stimare con buona approssimazione l'errore relativo sul risultato con il residuo pesato con il termine noto (i.e. il test d'arresto del residuo è affidabile).

I seguenti esempi mostrano che la convergenza del metodo di Gauss-Seidel non implica quella del metodo di Iacobi (1) e viceversa (2). Se entrambi convergono, il numero di iterazioni del metodo di Gauss-Seidel non è confrontabile a priori con quello delle iterazioni prodotte dal metodo di Iacobi (3)(4).

Per maggior chiarezza i risultati sono riassunti nella tabella seguente

	# iterazioni						$\rho(P)$		$\|P\|_\infty$		K(A)
	I	GS	I	GS	I	GS	I	GS	I	GS	
1	100	27	100	27	100	26	1.1	0.5	2	1	1.9
2	4	100	4	100	4	100	$\sim 10^{-5}$	4.8	4	14	36
3	17	6	17	6	16	6	0.4	0.02	1.33	0.5	4
4	35	57	35	61	32	53	0.6	0.8	2.14	2.51	5.9

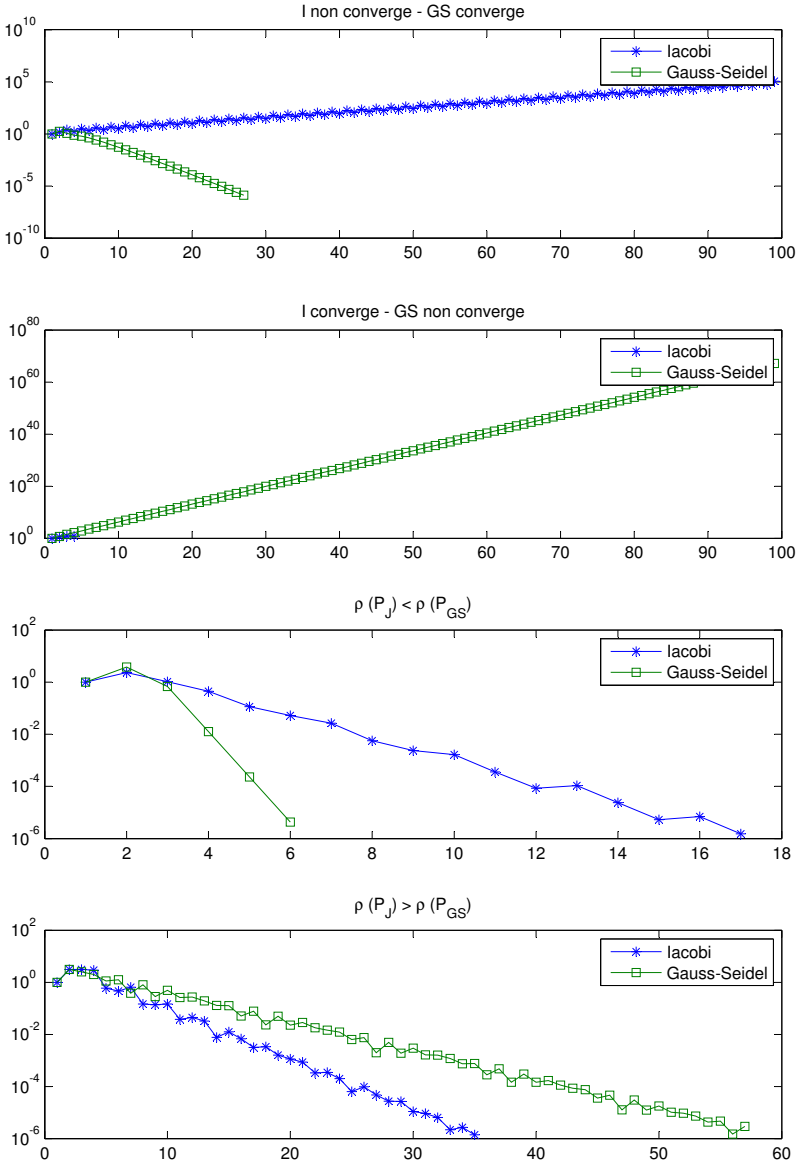

Figura 21: *I due metodi non hanno in generale lo stesso carattere (convergenza) e, se convergono, non è possibile confrontarne a priori le velocità di convergenza.*

Confrontare i risultati ottenuti con il test d'arresto dell'incremento (**norma 2**)

```
 SISTEMA: Ax=b

     2    -1     1
     2     2     2
    -1    -1     2

Soluzioni: IACOBI [#iterazioni: 100]
norma della matrice di iterazione: 1.500000e+000
Soluzioni: GAUSS-SEIDEL [#iterazioni: 27]
norma della matrice di iterazione: 8.660254e-001
-----------------------------------------------------------------
SISTEMA: Ax=b

     1    -2     2
    -1     1    -1
    -2    -2     1

Soluzioni: IACOBI [#iterazioni: 4]
norma della matrice di iterazione: 3.464102e+000
Soluzioni: GAUSS-SEIDEL [#iterazioni: 100]
norma della matrice di iterazione: 1.061511e+001
-----------------------------------------------------------------
SISTEMA: Ax=b

     4     1     1
     2    -9     0
     0    -8    -6

Soluzioni: IACOBI [#iterazioni: 17]
norma della matrice di iterazione: 1.357377e+000
Soluzioni: GAUSS-SEIDEL [#iterazioni: 6]
norma della matrice di iterazione: 3.770236e-001
-----------------------------------------------------------------
SISTEMA: Ax=b

     7     6     9
     4     5    -4
    -7    -3     8

Soluzioni: IACOBI [#iterazioni: 35]
norma della matrice di iterazione: 1.733731e+000
Soluzioni: GAUSS-SEIDEL [#iterazioni: 61]
norma della matrice di iterazione: 2.549433e+000
-----------------------------------------------------------------
```

Confrontare i risultati ottenuti con il **test d'arresto del residuo**

```
SISTEMA: Ax=b

     2    -1     1
     2     2     2
    -1    -1     2

Soluzioni: IACOBI [#iterazioni: 100]
Soluzioni: GAUSS-SEIDEL [#iterazioni: 26]
K(A): 1.958655e+000
-----------------------------------------------------------------
SISTEMA: Ax=b

     1    -2     2
    -1     1    -1
    -2    -2     1

Soluzioni: IACOBI [#iterazioni: 4]
Soluzioni: GAUSS-SEIDEL [#iterazioni: 100]
K(A): 3.688094e+001
-----------------------------------------------------------------
SISTEMA: Ax=b

     4     1     1
     2    -9     0
     0    -8    -6

Soluzioni: IACOBI [#iterazioni: 16]
Soluzioni: GAUSS-SEIDEL [#iterazioni: 6]
K(A): 4.089535e+000
-----------------------------------------------------------------
SISTEMA: Ax=b

     7     6     9
     4     5    -4
    -7    -3     8

Soluzioni: IACOBI [#iterazioni: 32]
Soluzioni: GAUSS-SEIDEL [#iterazioni: 53]
K(A): 5.977110e+000
-----------------------------------------------------------------
```

10.5 Metodo di Rilassamento SOR

```
function [x,k,r,nP] = sor(A,b,tol,Nmax,xo,omega)
% SOR Metodo iterativo di rilassamento sor
%     per il calcolo della soluzione del sistema Ax=b
% Dati input:
%  A     matrice quadrata dei coefficienti del sistema
%  b     termine noto del sistema
%  tol   tolleranza
%  Nmax  numero massimo di iterazioni
%  xo  vettore d'innesco
%  omega parametro di rilassamento
%
% Dati output:
%  x     soluzione del sistema
%  k     numero di iterazioni compiute
%  r     raggio spettrale della matrice di iterazione
%  nP    norma inf della matrice di iterazione

n = size(A,2);
x = xo;
err = 1;
k = 0;

L = A-triu(A);
U = A-tril(A);
D = diag(diag(A));
P = inv(D+omega*L)*((1-omega)*D-omega*U);
r = max(abs(eig(P)));
nP = norm(P,inf);
while err>tol && k<Nmax
    x(1)=omega*(b(1)-A(1,2:n)*x(2:n))/A(1,1)+(1-omega)*x(1);
    for i=2:n-1
        x(i)=omega*(b(i)-A(i,1:i-1)*x(1:i-1)-A(i,i+1:n)...
        *x(i+1:n))/A(i,i)+(1-omega)*x(i);
    end
    x(n)=omega*(b(n)-A(n,1:n-1)*x(1:n-1))/A(n,n)+(1-omega)*x(n);
    err = norm(x-xo,inf);
    k = k+1;
    xo = x;
end
end
```

10.6 *esercizio 7*

Confrontare i metodi di eliminazione di Gauss attraverso fattorizzazione LU con strategia pivotale parziale, i metodi iterativi di Iacobi e Gauss-Seidel e di rilassamento SOR al variare del coefficiente ω.

```
n=6;
tol=10^(-4);
Nmax=100;
xo=zeros(1,n)';

e = ones(n,1);
b = [-e -e 4*e -e -e];
d = [-n/2 -1 0 1 n/2];
A = full(spdiags(b,d,n,n));
A(4,3)=0;
A(3,4)=0;
b = [2 1 2 2 1 2]';
metodi_iterativi_stampa_sor(A,b,tol,Nmax,xo);

function [] = metodi_iterativi_stampa_sor(A,b,tol,Nmax,xo)

[x1,k1,r1,nP1] = iacobi(A,b,tol,Nmax,xo);
[x2,k2,r2,nP2] = gseidel(A,b,tol,Nmax,xo);

disp('SISTEMA: Ax=b')
A
b=b'
fprintf('Soluzioni: GAUSS PIVOT PARZIALE')
x=(A\b')
%.............................................................
fprintf('\nSoluzioni: IACOBI [#iterazioni: %d]',k1)
x1
err = abs(norm(x1)-norm(ones(1,6)))/norm(ones(1,6));
fprintf('errore relativo sulla soluzione (norma 2): %.6f\n',err)
fprintf('raggio spettrale della matrice di iterazione: %.6f\n',r1)
fprintf('norma inf della matrice di iterazione: %.6f\n',nP1)
%.............................................................
fprintf('\nSoluzioni: GAUSS-SEIDEL [#iterazioni: %d]',k2)
x2
err = abs(norm(x2)-norm(ones(1,6)))/norm(ones(1,6));
fprintf('errore relativo sulla soluzione (norma 2): %.6f\n',err)
fprintf('raggio spettrale della matrice di iterazione: %.6f\n',r2)
fprintf('norma inf della matrice di iterazione: %.6f\n',nP2)
%.............................................................
for k=[0:5]
    omega=1+0.2*k;
```

```
    [x3,k3,r3,nP3] = sor(A,b,tol,Nmax,xo,omega);
    fprintf('\nSoluzioni: RILASSAMENTO SOR ...
     [#iterazioni: %d, omega: %.2f]',k3,omega)
    x3
    err = abs(norm(x3)-norm(ones(1,6)))/norm(ones(1,6));
    fprintf('errore relativo sulla soluzione (norma 2): %.6f\n',err)
    fprintf('raggio spettrale della matr. di iterazione: %.6f\n',r3)
    fprintf('norma della matrice di iterazione: %.6f\n',nP3)
end
fprintf('\n\nTOLLERANZA: %s; #ITERAZIONI MASSIME: %d\n',tol,Nmax)
disp('------------------------------------------------------------')
end
```

```
SISTEMA: Ax=b

A =
     4    -1     0    -1     0     0
    -1     4    -1     0    -1     0
     0    -1     4     0     0    -1
    -1     0     0     4    -1     0
     0    -1     0    -1     4    -1
     0     0    -1     0    -1     4
b =
     2     1     2     2     1     2

Soluzioni: GAUSS PIVOT PARZIALE
x =
    1.000000000000000e+000
    9.999999999999999e-001
    1.000000000000000e+000
    9.999999999999999e-001
    1.000000000000000e+000
    1.000000000000000e+000

Soluzioni: IACOBI [#iterazioni: 18]
x1 =
    9.999035919754533e-001
    9.998636584641645e-001
    9.999035919754533e-001
    9.999035919754533e-001
    9.998636584641645e-001
    9.999035919754533e-001

errore relativo sulla soluzione (norma 2): 0.000110
raggio spettrale della matrice di iterazione: 0.603553
```

```
norma inf della matrice di iterazione: 0.750000

Soluzioni: GAUSS-SEIDEL [#iterazioni: 11]
x2 =
    9.999615025458297e-001
    9.999671403664934e-001
    9.999859762741323e-001
    9.999767647306044e-001
    9.999801674567834e-001
    9.999915359327289e-001

errore relativo sulla soluzione (norma 2): 0.000023
raggio spettrale della matrice di iterazione: 0.364277
norma inf della matrice di iterazione: 0.625000

Soluzioni: RILASSAMENTO SOR [#iterazioni: 11, omega: 1.00]
x3 =
    9.999615025458297e-001
    9.999671403664934e-001
    9.999859762741323e-001
    9.999767647306044e-001
    9.999801674567834e-001
    9.999915359327289e-001

errore relativo sulla soluzione (norma 2): 0.000023
raggio spettrale della matrice di iterazione: 0.364277
norma della matrice di iterazione: 0.625000

Soluzioni: RILASSAMENTO SOR [#iterazioni: 8, omega: 1.20]
x3 =
    9.999928384770180e-001
    1.000006632325860e+000
    1.000000176517432e+000
    1.000002837425945e+000
    1.000004724971149e+000
    1.000000769169060e+000

errore relativo sulla soluzione (norma 2): 0.000001
raggio spettrale della matrice di iterazione: 0.200000
norma della matrice di iterazione: 0.860000

Soluzioni: RILASSAMENTO SOR [#iterazioni: 13, omega: 1.40]
x3 =
    9.999915296319328e-001
    9.999995599396526e-001
    1.000002002949795e+000
```

```
    1.000011219983852e+000
    1.000005391500757e+000
    1.000004368422719e+000

errore relativo sulla soluzione (norma 2): 0.000002
raggio spettrale della matrice di iterazione: 0.400000
norma della matrice di iterazione: 1.240000

Soluzioni: RILASSAMENTO SOR [#iterazioni: 21, omega: 1.60]
x3 =
    1.000021945492680e+000
    9.999797654871184e-001
    9.999964928028223e-001
    9.999752384014422e-001
    9.999839648958272e-001
    9.999878124566121e-001

errore relativo sulla soluzione (norma 2): 0.000009
raggio spettrale della matrice di iterazione: 0.600000
norma della matrice di iterazione: 1.640000

Soluzioni: RILASSAMENTO SOR [#iterazioni: 46, omega: 1.80]
x3 =
    9.999840780957710e-001
    1.000026195067577e+000
    1.000001360253645e+000
    1.000034156012066e+000
    1.000034519783288e+000
    1.000030658263463e+000

errore relativo sulla soluzione (norma 2): 0.000018
raggio spettrale della matrice di iterazione: 0.800000
norma della matrice di iterazione: 2.060000

Soluzioni: RILASSAMENTO SOR [#iterazioni: 100, omega: 2.00]
x3 =
    1.104084005496704e-001
    1.197294730564465e+000
    1.187735696695456e+000
    6.361192788236190e-001
    2.104806896087694e+000
    7.014767553380081e-001

errore relativo sulla soluzione (norma 2): 0.167858
raggio spettrale della matrice di iterazione: 1.000000
norma della matrice di iterazione: 2.500000
```

Osservazione

Il metodo di I (18 iterazioni, $\rho(P) = 0.60$) converge più lentamente del metodo di GS (11 iterazioni, $\rho(P) = 0.36$); il metodo SOR per $\omega = 1.2$ converge in un numero di iterazioni ancora minore (8, $\rho(P) = 0.2$). Confrontando gli errori relativi sulla soluzione il metodo SOR $\omega = 1.2$ è anche il più stabile. La convergenza del metodo SOR è garantita all'interno dell'intervallo aperto $(0, 2)$: per $\omega = 2$ il metodo non converge.

10.7 *esercizio 8*

Confrontare i metodi iterativi di rilassamento SOR al variare del coefficiente ω.

```
m = 10;
A = diag(2*ones(m,1),0)+diag(-ones(m-1,1),1)+diag(-ones(m-1,1),-1);
x = ones(m,1);
b = A*x;
tol=10^(-6);
Nmax=100;
xo=zeros(m,1);

e = [0.1:0.1:1.9];
d = zeros(max(size(e)),1);
k = d;
r = d;
nP = d;
j = 1;
p = 0.1;
errore = zeros(100,max(size(e)));
count = 1;
for i = [0.1:p:1.9]
   [x,k(j),r(j),nP(j),err] = sor_err(A,b,tol,Nmax,xo,i);
   errore(1:max(size(err)),j) = err;
   j=j+1;
end

for i = 1:size(errore,1)
    for j = 1:size(errore,2)
        if errore(i,j)==0 || i==1
           errore(i,j)=NaN;
        end
    end
end

omega = [0.1:p:1.9]';
disp('    omega   #iteration  r.spett   norm(P)');
disp([omega,k,r,nP])
```

```
figure;
subplot(3,1,1)
plot([0.1:p:1.9],k')
title('#iterazioni')
subplot(3,1,2)
plot([0.1:p:1.9],r')
title('raggio spettrale della matrice di iterazione')
subplot(3,1,3)
plot([0.1:p:1.9],nP')
title('norma della matrice di iterazione')
figure;
mesh(errore,'DisplayName','errore');
figure(gcf)
```

omega	#iteration	r.spett	norm(P)
0.1000	100.0000	0.9957	1.0000
0.2000	100.0000	0.9910	1.0000
0.3000	100.0000	0.9858	1.0000
0.4000	100.0000	0.9799	1.0000
0.5000	100.0000	0.9733	1.0000
0.6000	100.0000	0.9658	1.0000
0.7000	100.0000	0.9571	0.9999
0.8000	100.0000	0.9470	0.9997
0.9000	100.0000	0.9350	0.9992
1.0000	100.0000	0.9206	0.9980
1.1000	100.0000	0.9029	0.9971
1.2000	98.0000	0.8803	0.9966
1.3000	79.0000	0.8500	0.9984
1.4000	63.0000	0.8059	1.1000
1.5000	46.0000	0.7280	1.2500
1.6000	35.0000	0.6000	1.4000
1.7000	46.0000	0.7000	1.5500
1.8000	69.0000	0.8000	1.7000
1.9000	100.0000	0.9000	1.8500

raggio spettrale	norma(P)
9.957449449772183e-001	9.999999999980469e-001
9.910366297720344e-001	9.999999990000000e-001
9.857963333667121e-001	9.999999615566406e-001
9.799253654032124e-001	9.999994880000001e-001
9.732980709350882e-001	9.999961853027344e-001
9.657515565769155e-001	9.999803169999999e-001
9.570701183886947e-001	9.999211843613280e-001
9.469607305511458e-001	9.997378560000000e-001
9.350126381934376e-001	9.992433193574219e-001

```
9.206267664155903e-001    9.980468750000000e-001
9.028827521531411e-001    9.970693121738281e-001
8.802615014790604e-001    9.966407680000000e-001
8.499735927094967e-001    9.984067759355469e-001
8.058902655493139e-001    1.100000000000000e+000
7.280068731455316e-001    1.250000000000000e+000
5.999999999999996e-001    1.400000000000000e+000
7.000000000000005e-001    1.550000000000000e+000
8.000000000000007e-001    1.700000000000000e+000
9.000000000000014e-001    1.850000000000000e+000
```

Osservazione

Il metodo di I (263 iterazioni, $\rho(P) = 0.95$) e il metodo di GS (141 iterazioni, $\rho(P) = 0.92$) convergono in un numero di iterazioni maggiore del numero massimo, preliminarmente fissato (100); il metodo SOR per $\omega = 1.6$ converge dopo 35 iterazioni.

$$\omega_{opt} = \frac{2}{1 + \sqrt{1 - \rho(P_j)^2}} \sim 1.560387921274774$$

Il valore teorico è coerente con i risultati ottenuti. Scegliamo di visualizzare gli errori assoluti per ω convergenti attraverso la function `mesh`. La proiezione del grafico sul piano xy è il sottografico della funzione $\#iterazioni(\omega)$. L'intersezione con piani verticali paralleli al piano xz dà ragione della diversa velocità di convergenza del metodo SOR al variare di ω e richiama il grafico del $\rho(P)(\omega)$. Le irregolarità sulla superficie (la non-monotonia delle intersezioni del grafico con piani paralleli al piano yz) sono dovute al fatto che per opportuni ω $\|P\|_\infty > 1$

```
Soluzioni: IACOBI [#iterazioni: 263]
- errore relativo sulla soluzione (norma 2): 0.290973
- raggio spettrale della matrice di iterazione: 0.959493
- norma inf della matrice di iterazione: 1.000000

Soluzioni: GAUSS-SEIDEL [#iterazioni: 141]
- errore relativo sulla soluzione (norma 2): 0.290984
- raggio spettrale della matrice di iterazione: 0.920627
- norma inf della matrice di iterazione: 0.998047

Soluzioni: RILASSAMENTO SOR [#iterazioni: 35, omega: 1.60]
- raggio spettrale della matrice di iterazione: 0.600000
- norma della matrice di iterazione: 1.40000000
```

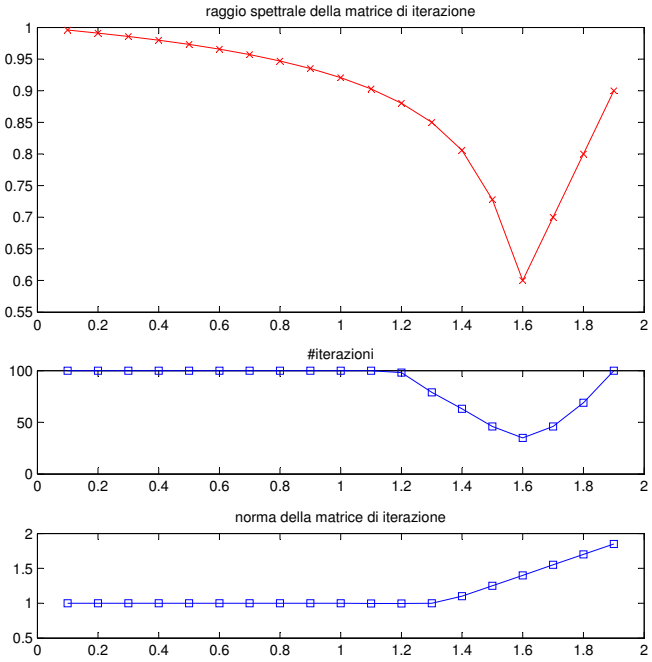
raggio spettrale della matrice di iterazione
1
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2
#iterazioni
100
50
0
0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2
norma della matrice di iterazione
2
1.5
1
0.5
0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2

Figura 22

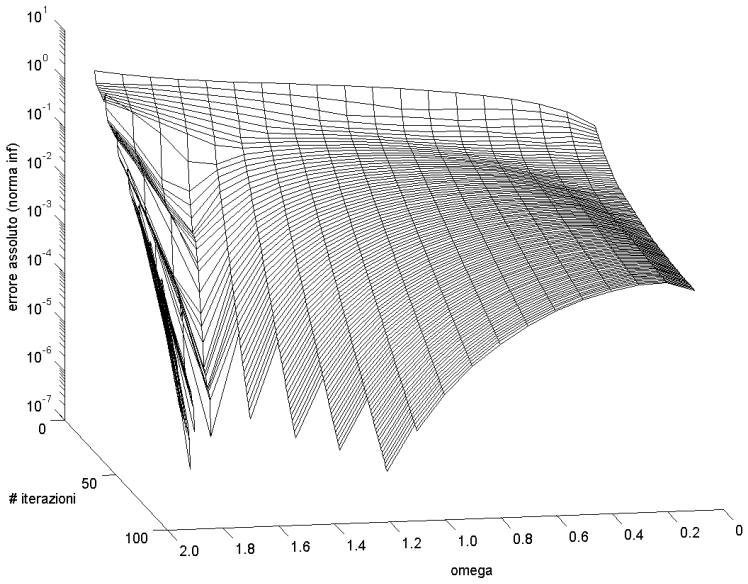

Figura 23: errore assoluto: `asse x = omega; asse y = # iterazioni; asse z = errore assoluto (norma inf)`. La depressione in corrispondenza di $\omega_{opt} \sim 1.5604$ segnala la massima velocità di convergenza del SOR

Conclusioni

- Il metodo di eliminazione di Gauss con fattorizzazione LU e strategia pivotale parziale rappresenta il miglior compromesso tra stabilità e costo computazionale; pertanto è il metodo implementato dalla libreria `LINPACK`.
- Il metodo Cholesky è una variante del metodo di eliminazione di Gauss come sopra per matrici simmetriche definite positive (adottato in alcune strategie di precondizionamento per il contenimento dell'effetto di fill-in).
- Il metodo QR assicura una maggior stabilità numerica, a svantaggio dei tempi di calcolo; utilizzato per problemi che richiedono una grande precisione nei risultati (e.g. calcolo degli autovalori di un'applicazione lineare)
- I metodi iterativi conservano la struttura della matrice, poiché non la modificano ad ogni passo di iterazione (particolarmente indicati per matrici sparse). In generale sono più stabili dei metodi diretti poiché, valutando le iterate nella stessa matrice assegnata, i coefficienti di amplificazione dell'errore algoritmico ad ogni iterazione sono nulli. Tuttavia, si introducono problemi di convergenza, di determinazione dei test d'arresto e di eventuali parametri di ottimizzazione.

Parte III
AUTOVALORI DI MATRICI

11 METODO DELLE POTENZE

11.1 Autovalore di modulo massimo

```
function [y,lambda,k] = potenza2(A,y,Maxit,tol)
% POTENZA2 metodo delle potenze per il calcolo dell'autovalore di modulo
%          massimo
y = y/norm(y);
k = 1;
err = 1;
sigma0 = 1;
while k <= Maxit && err > tol
    z = A*y;
    sigma = (y')*z;
    if sigma == 0
        break % per matrici definite positive sigma > 0
    end
    err = abs(sigma-sigma0);
    y = z/norm(z);
    sigma0 = sigma;
    k = k+1;
end
lambda = sigma;
if k == Maxit
    fprintf('non converge in %d iterazioni',Maxit);
end
end
```

11.2 Metodo delle potenze con spostamento

```
function [y,lambda,k] = potenza_inverse2(A,y,Maxit,tol,a)
% POTENZA_INVERSE2 metodo delle potenze inverse per il calcolo
%                  dell'autovalore di modulo minimo (a = 0) o la stima
%                  dell'autovalore prossimo ad (a ~= 0)
n = max(size(A));
y = y/norm(y);
k = 1;
sigma0 = 1;
A = A - a*eye(n);
[L,U,P]=lu(A);
k = 1;
err = 1;
```

```
sigma0 = 1;
while k <= Maxit && err > tol
    z = U\(L\(P*y));
    sigma = y'*z;
    if sigma == 0
        break % per matrici definite positive sigma > 0
    end
    err = abs(sigma-sigma0);
    y = z/norm(z);
    sigma0 = sigma;
    k = k+1;
end
lambda = 1/sigma+a;
if k == Maxit
    fprintf('non converge in %d iterazioni',Maxit);
end
end
```

11.3 *esercizio 1*

Applicare il metodo delle potenze alle seguenti matrici per il calcolo dell'autovalore di modulo massimo. Implementare la function `ratioeig` che calcoli l'inverso del rapporto tra l'autovalore di modulo massimo e quello immediatamente successivo rispetto tale relazione d'ordine (che nel seguito saranno rispettivamente denotati λ_1 e λ_2). Tale rapporto è misura della velocità di convergenza del metodo delle potenze; vale infatti per l'iterata k-esima la seguente relazione:

$$\sigma_k = \lambda_1 \left(1 + O\left(\left| \frac{\lambda_{r+1}}{\lambda_1} \right|^k \right) \right)$$

e vale con l'esponente $2k$ per matrici hermetiane

```
A1=[-7 -9 9 ;11 13 -9;-16 -16, 20];
A2=[ -4 -5 4 ; 14 15 -5; -1 -1 11];
A3=[5 3.2 7; 8.3 5 4.2; 0.1 3.3 -10];
A4=[1 2 3 4; 2 3 4 0; 3 4 1 2; 4 0 2 3];
A5=[-3 2 -2 4; 6 1 2 -4 ;6 -2 5 -4 ;-6 2 -2 7 ];

tol=10^(-6);
Maxit=1500;

disp('A1:')
potenza2_stampa(A1,Maxit,tol)
disp('A2:')
potenza2_stampa(A2,Maxit,tol)
disp('A3:')
```

```
potenza2_stampa(A3,Maxit,tol)
disp('A4:')
potenza2_stampa(A4,Maxit,tol)

tol=10^(-16);
disp('A5:')
potenza2_stampa(A5,Maxit,tol)
fprintf('N.B. autovalore max A5 appartiene a autospazio di
... autovalore max:\n')
[V,D] = eig(A5);
y = ones(max(size(A5)),1);
[y,lambda,k] = potenza2(A5,y,Maxit,tol);
V(:,1) = y;
disp('A = [aut max| autospazio]: ')
disp(V)
fprintf('rango A: %d\n',rank(V))


function [] = potenza2_stampa(A,Maxit,tol)
% POTENZE2_STAMPA stampa output metodo delle potenze
    y = ones(max(size(A)),1);
    disp(A)
    fprintf('spettro:\n')
    [V,D] = eig(A);
    disp(diag(D)')
    fprintf('autovettori corrispondenti:\n')
    disp(V)
    fprintf('METODO DELLE POTENZE\n')
    [y,lambda,k] = potenza2(A,y,Maxit,tol);
    fprintf('autovalore di modulo max [#iterazioni: %d]:
    ... %d\n',k,lambda)
    l = ratioeig(A);
    fprintf('ratioeig: %.6f\n',l)
    fprintf('autovettore corrispondente:\n')
    disp(y)
    fprintf('-------------------------------------------------\n')
end


function [l] = ratioeig(A)
%RATIOEIG Calcola il rapporto tra l'autovalore
%         degli autovalori di modulo maggiore
    [V,D]=eig(A);
    D = abs(diag(D));
    lambda1 = max(D);
    for k = 1: max(size(A))
```

```
        if abs(lambda1 - D(k)) < 10^2*eps
           D(k) = 0;
        end
    end
    lambda2 = max(D);
    l = lambda2/lambda1;
end
```

```
A1:
    -7    -9     9
    11    13    -9
   -16   -16    20

spettro:
  2.0000e+001  2.0000e+000  4.0000e+000

autovettori corrispondenti:
  4.0825e-001 -7.0711e-001  9.4369e-016
 -4.0825e-001  7.0711e-001  7.0711e-001
  8.1650e-001  1.0878e-015  7.0711e-001

METODO DELLE POTENZE
autovalore di modulo max [#iterazioni: 12]: 2.000000e+001
ratioeig: 0.200000
autovettore corrispondente:
 -4.0825e-001
  4.0825e-001
 -8.1650e-001
```

```
A2:
    -4    -5     4
    14    15    -5
    -1    -1    11

spettro:
  1.0000e+000  1.1618e+001  9.3820e+000

autovettori corrispondenti:
  7.0711e-001  3.7931e-001 -2.0487e-001
 -7.0711e-001 -7.3038e-001  8.8418e-001
 -2.2425e-017  5.6804e-001  4.1984e-001

METODO DELLE POTENZE
autovalore di modulo max [#iterazioni: 70]: 1.161803e+001
ratioeig: 0.807535
```

```
autovettore corrispondente:
 -3.7931e-001
  7.3038e-001
 -5.6804e-001

------------------------------------------------------------
A3:
  5.0000e+000  3.2000e+000  7.0000e+000
  8.3000e+000  5.0000e+000  4.2000e+000
  1.0000e-001  3.3000e+000 -1.0000e+001

spettro:
  1.1319e+001 -1.1844e+000 -1.0135e+001

autovettori corrispondenti:
 -5.5836e-001 -6.5774e-001 -4.1538e-001
 -8.1943e-001  7.0786e-001 -2.4545e-002
 -1.2946e-001  2.5752e-001  9.0932e-001

METODO DELLE POTENZE
autovalore di modulo max [#iterazioni: 138]: 1.131918e+001
ratioeig: 0.895361
autovettore corrispondente:
  5.5836e-001
  8.1943e-001
  1.2946e-001

------------------------------------------------------------
A4:
     1     2     3     4
     2     3     4     0
     3     4     1     2
     4     0     2     3

spettro:
 -2.7016e+000 -2.5208e+000  3.7016e+000  9.5208e+000

autovettori corrispondenti:
  6.6725e-001 -4.7879e-001  2.3406e-001  5.2035e-001
  2.3406e-001  5.2035e-001 -6.6725e-001  4.7879e-001
 -6.6725e-001 -4.7879e-001 -2.3406e-001  5.2035e-001
 -2.3406e-001  5.2035e-001  6.6725e-001  4.7879e-001

METODO DELLE POTENZE
autovalore di modulo max [#iterazioni: 6]: 9.520797e+000
ratioeig: 0.388787
```

```
autovettore corrispondente:
  5.2034e-001
  4.7880e-001
  5.2034e-001
  4.7880e-001

------------------------------------------------------------

A5:
    -3     2    -2     4
     6     1     2    -4
     6    -2     5    -4
    -6     2    -2     7

spettro:
  1.0000e+000  3.0000e+000  3.0000e+000  3.0000e+000

autovettori corrispondenti:
  5.0000e-001 -4.8564e-001 -3.1919e-001  4.7948e-001
 -5.0000e-001  2.9139e-001 -9.4768e-001 -1.4331e-001
 -5.0000e-001  5.8277e-001  3.2934e-003  1.3013e-001
  5.0000e-001 -5.8277e-001 -3.2934e-003  8.5594e-001

METODO DELLE POTENZE
autovalore di modulo max [#iterazioni: 34]: 3
ratioeig: 0.333333
autovettore corrispondente:
 -2.7590e-016
  7.0711e-001
  7.0711e-001
 -9.4243e-017

------------------------------------------------------------

N.B. autovalore max A5 appartiene a autospazio di autovalore max:
A = [aut max| autospazio]:
 -2.7590e-016 -4.8564e-001 -3.1919e-001  4.7948e-001
  7.0711e-001  2.9139e-001 -9.4768e-001 -1.4331e-001
  7.0711e-001  5.8277e-001  3.2934e-003  1.3013e-001
 -9.4243e-017 -5.8277e-001 -3.2934e-003  8.5594e-001

rango A: 3
```

Osservazioni

A2 il metodo converge in un numero elevato di iterazioni (70); infatti

$$\left|\frac{\lambda_2}{\lambda_1}\right| = 0.8075347361124052 \sim 1$$

A3 il metodo converge in un numero elevato di iterazioni (139); infatti

$$\left|\frac{\lambda_2}{\lambda_1}\right| = 0.8953611289497810 \sim 1$$

A4 imposta la stessa tolleranza (10^{-6}) e lo stesso numero massimo di iterazioni (1500), e sia

$$\left|\frac{\lambda_{2,A4}}{\lambda_{1,A4}}\right| \sim \left|\frac{\lambda_{2,A5}}{\lambda_{1,A5}}\right|$$

la velocità di convergenza del metodo (i.e. il numero di iterazioni) per matrici simmetriche è maggiore della velocità di convergenza per matrici generiche

```
A4:
     1     2     3     4
     2     3     4     0
     3     4     1     2
     4     0     2     3

METODO DELLE POTENZE
autovalore di modulo max [#iterazioni: 6]: 9.520797e+000
ratioeig: 0.388787

-------------------------------------------------------------
A5:
    -3     2    -2     4
     6     1     2    -4
     6    -2     5    -4
    -6     2    -2     7

METODO DELLE POTENZE
autovalore di modulo max [#iterazioni: 15]: 3.000000e+000
ratioeig: 0.333333
```

A5 l'autovettore calcolato con il metodo delle potenze appartiene all'autospazio dell'autovalore di modulo massimo calcolato dalla function `eig`. A patto di abbassare la tolleranza (e di conseguenza il numero massimo di iterazioni consentite: `tol=10`$^{-16}$`;Maxit=1500`), il rango della matrice costruita allineando l'autovalore calcolato e una base dell'autospazio è massimo. Se la molteplicità dell'autovalore di modulo massimo è maggiore di 1, i.e.

$\exists r \in \mathbb{N} \,|\, \lambda_1 = \ldots = \lambda_r$ *autovalori t.c.* $|\lambda_1| > |\lambda_i| \ \forall i > r$, il metodo converge, e la sua velocità di convergenza è pari a

$$\left| \frac{\lambda_{r+1}}{\lambda_1} \right|$$

Nota La stabilità della function `ratioeig` è discutibile: è implementata attraverso la function `eig` e, per il metodo delle potenze con spostamento, attraverso la function `inv`. Per evitare risultati incoerenti, valuta due autovalori coincidenti se distano per meno di $100\,eps \sim 2.220446049250313e-014$. Ciononostante, la stima è sufficiente per le osservazioni seguenti.

11.4 *esercizio 2*

Osservare patologie e limiti del metodo delle potenze

```
B1=[2 1 3 4; 1 -3 1 5; 3 1 6 -2; 4 5 -2 -1];
B2=[ -1 1 -1; 5 3 1 ; 8 8 -4];
B3=[4 141 -576 432;1 0 0 0 ; 0 1 0 0; 0 0 1 0];

tol=10^(-6);
Maxit=1500;

% metodo delle potenze B1
    y = ones(max(size(B1)),1);
    disp('B1:')
    disp(B1)
    fprintf('spettro:\n')
    [V,D] = eig(B1);
    disp(diag(D)')
    fprintf('autovettori corrispondenti:\n')
    disp(V)
    fprintf('METODO DELLE POTENZE\n')
    [y,lambda,k] = potenza2(B1,y,Maxit,tol);
    fprintf('autovalore di modulo max [#iterazioni: %d]:
    ...%d\n',k,lambda)
    l = ratioeig(B1);
    fprintf('ratioeig: %.6f\n',l)
    fprintf('autovettore corrispondente:\n')
    disp(y)
    y = ones(max(size(B1)),1);
    a = -6;
    [y,lambda,k] = potenza_inverse2(B1,y,Maxit,tol,a);
    fprintf('METODO DELLE POTENZE CON SPOSTAMENTO
    ... [shift: %d]\n',a)
    fprintf('autovalore [#iterazioni: %d]: %d\n',k,lambda)
    fprintf('autovettore corrispondente:\n')
```

```
    disp(y)
    disp('-------------------------------------------------')
% metodo delle potenze B2 e B3
    disp('B2:')
    potenza2_stampa(B2,Maxit,tol)
    disp('B3:')
    potenza2_stampa(B3,Maxit,tol)
    fprintf('N.B. vettore innesco = [1,2,3]: \n')
    y = [1:max(size(B3))]';
    [y,lambda,k] = potenza2(B3,y,Maxit,tol);
    fprintf('autovalore di modulo max [#iterazioni:
    ...%d]: %d\n',k,lambda)
    fprintf('autovettore corrispondente:\n')
    disp(y)
```

```
B1:
     2     1     3     4
     1    -3     1     5
     3     1     6    -2
     4     5    -2    -1

spettro:
 -8.0286e+000 -1.5732e+000  5.6689e+000  7.9329e+000

autovettori corrispondenti:
  2.6346e-001 -6.8805e-001  3.7870e-001  5.6014e-001
  6.5904e-001  6.2412e-001  3.6242e-001  2.1163e-001
 -1.9963e-001  2.5980e-001 -5.3794e-001  7.7671e-001
 -6.7557e-001  2.6375e-001  6.6020e-001  1.9538e-001

METODO DELLE POTENZE
autovalore di modulo max [#iterazioni: 839]: -8.028538e+000
ratioeig: 0.988083
autovettore corrispondente:
 -2.6258e-001
 -6.5871e-001
  2.0086e-001
  6.7588e-001

METODO DELLE POTENZE CON SPOSTAMENTO [shift: -6]
autovalore [#iterazioni: 14]: -8.028579e+000
autovettore corrispondente:
  2.6334e-001
  6.5915e-001
 -1.9959e-001
 -6.7553e-001
```

```
------------------------------------------------------------
B2:
    -1     1    -1
     5     3     1
     8     8    -4

spettro:
  4.0000e+000 -2.0000e+000 -4.0000e+000

autovettori corrispondenti:
 -4.9285e-017 -7.0711e-001  4.0825e-001
 -7.0711e-001  7.0711e-001 -4.0825e-001
 -7.0711e-001  6.4355e-016  8.1650e-001

METODO DELLE POTENZE
--> non converge in 1500 iterazioni
autovalore di modulo max [#iterazioni: 1500]: 2.260870e+000
ratioeig: 0.500000
autovettore corrispondente:
 -1.8257e-001
  9.1287e-001
  3.6515e-001

------------------------------------------------------------
B3:
     4   141  -576   432
     1     0     0     0
     0     1     0     0
     0     0     1     0

spettro:
 -1.2000e+001  1.2000e+001  3.0000e+000  1.0000e+000

autovettori corrispondenti:
  9.9652e-001  9.9652e-001 -9.4288e-001  5.0000e-001
 -8.3043e-002  8.3043e-002 -3.1429e-001  5.0000e-001
  6.9203e-003  6.9203e-003 -1.0476e-001  5.0000e-001
 -5.7669e-004  5.7669e-004 -3.4922e-002  5.0000e-001

METODO DELLE POTENZE
autovalore di modulo max [#iterazioni: 1]: 1
ratioeig: 0.250000
autovettore corrispondente:
  5.0000e-001
  5.0000e-001
```

```
  5.0000e-001
  5.0000e-001

------------------------------------------------------------
N.B. vettore innesco = [1,2,3,4]:
--> non converge in 1500 iterazioni
autovalore di modulo max [#iterazioni: 1500]: 4.816005e+000
autovettore corrispondente:
  9.7886e-001
  2.0444e-001
  6.7976e-003
  1.4197e-003
```

Osservazioni

B1 B1 converge molto lentamente (`ratioeig` ~ 1). Il metodo delle potenze con spostamento può accelerare la convergenza (a patto di conoscere l'opportuno parametro di shift).

B2 il metodo delle potenze non converge poichè esistono due autovalori distinti di modulo massimo.

B3 il metodo si interrompe dopo la prima iterazione poichè il vettore d'innesco appartiene all'autospazio del autovalore di modulo massimo.

$$y = \begin{bmatrix}1\\1\\1\\1\end{bmatrix} \in V_{\lambda_1} = Span\left\{\begin{bmatrix}0.5\\0.5\\0.5\\0.5\end{bmatrix}\right\}$$

Al primo passo y è normalizzato ($\|y\| = 1$; $\lambda_1 = 1$)

$$\sigma_k = y^H A y = y^H(\lambda_1 y) = 1$$

Dopo n passi ($n =$`sigmao`) il ciclo termina. Supponiamo che il vettore d'innesco non appartenga all'autospazio dell'autovalore di modulo massimo, allora il metodo delle potenze non converge poichè esistono due autovalori distinti di modulo massimo.

11.5 *esercizio 3*

Applicare il metodo delle potenze alla seguente matrice per il calcolo dell'autovalore di modulo massimo e minimo. Calcolare l'autovalore prossimo ad opportuni parametri (paramatri di shift) con il metodo delle potenze con spostamento.

```
A=[15 -2 2; 1 10 -3;-2 1 0];
```

```
tol=10^(-6);
Maxit=1500;
y = ones(max(size(A)),1);

disp('A:')
disp(A)
fprintf('spettro:\n')
[V,D] = eig(A);
disp(diag(D)')
fprintf('autovettori corrispondenti:\n')
disp(V)
[y,lambda,k] = potenza2(A,y,Maxit,tol);
fprintf('METODO DELLE POTENZE\n')
fprintf('autovalore di modulo max [#iterazioni: %d]: %d\n',k,lambda)
l = ratioeig(A);
fprintf('ratioeig: %.6f\n',l)
fprintf('cond(A): %3.6f\n',cond(A));
fprintf('autovettore corrispondente:\n')
disp(y)

for a = [12:15]
    y = ones(max(size(A)),1);
    [y,lambda,k] = potenza_inverse2(A,y,Maxit,tol,a);
    fprintf('METODO DELLE POTENZE CON SPOSTAMENTO [shift: %d]\n',a)
    fprintf('autovalore [#iterazioni: %d]: %d\n',k,lambda)
    l = ratioeig(inv(A-a*eye(max(size(A)))));
    fprintf('ratioeig: %.6f\n',l)
    fprintf('cond(inv(A-aI)): %3.6f\n',cond(inv(A-a*eye(max(size(A))))));
    fprintf('autovettore corrispondente:\n')
    disp(y)
end

y = ones(max(size(A)),1);
[y,lambda,k] = potenza_inverse2(A,y,Maxit,tol,0);
fprintf('METODO DELLE POTENZE INVERSE\n')
fprintf('autovalore di modulo min [#iterazioni: %d]: %d\n',k,lambda)
l = ratioeig(inv(A));
fprintf('ratioeig: %.6f\n',l)
fprintf('cond(inv(A)): %3.6f\n',cond(inv(A)));
fprintf('autovettore corrispondente:\n')
disp(y)

A:
    15    -2     2
     1    10    -3
    -2     1     0
```

```
spettro:
  5.1208e-001  1.4103e+001  1.0385e+001

autovettori corrispondenti:
 -8.8117e-002 -9.4359e-001  3.9293e-001
  3.0874e-001 -3.1169e-001  9.1948e-001
  9.4706e-001  1.1172e-001  1.2866e-002

METODO DELLE POTENZE
autovalore di modulo max [#iterazioni: 43]: 1.410255e+001
ratioeig: 0.736417
cond(A): 33.243094
autovettore corrispondente:
    9.435920410960773e-001
    3.116945006717176e-001
   -1.117165978319585e-001

METODO DELLE POTENZE CON SPOSTAMENTO [shift: 12]
autovalore [#iterazioni: 60]: 1.038536e+001
ratioeig: 0.767942
cond(inv(A-aI)): 17.022969
autovettore corrispondente:
    3.929290576757182e-001
    9.194787734232913e-001
    1.286626822633774e-002

METODO DELLE POTENZE CON SPOSTAMENTO [shift: 13]
autovalore [#iterazioni: 17]: 1.410256e+001
ratioeig: 0.421685
cond(inv(A-aI)): 21.992692
autovettore corrispondente:
    9.435922210579623e-001
    3.116939312865962e-001
   -1.117166664289997e-001

METODO DELLE POTENZE CON SPOSTAMENTO [shift: 14]
autovalore [#iterazioni: 7]: 1.410256e+001
ratioeig: 0.028372
cond(inv(A-aI)): 205.026989
autovettore corrispondente:
    9.435921888473653e-001
    3.116940331984550e-001
   -1.117166541511080e-001

METODO DELLE POTENZE CON SPOSTAMENTO [shift: 15]
autovalore [#iterazioni: 10]: 1.410256e+001
```

```
ratioeig: 0.194478
cond(inv(A-aI)): 22.500080
autovettore corrispondente:
    9.435921829319196e-001
    3.116940519147882e-001
   -1.117166518953923e-001

METODO DELLE POTENZE INVERSE
autovalore di modulo min [#iterazioni: 6]: 5.120848e-001
ratioeig: 0.049308
cond(inv(A)): 33.243094
autovettore corrispondente:
   -8.811725678187253e-002
    3.087386819609979e-001
    9.470563738860656e-001
```

Osservazioni

Quanto più il parametro di shift avvicina l'autovalore, tanto più aumenta la velocità di convergenza, ma anche il condizionamento della matrice $(A - \mu I)^{-1}$ (si avvicina ad una matrice singolare) con il rischio di perdita di accuratezza nei risultati.

11.6 *esercizio 4*

Applicare il metodo delle potenze con spostamento alla seguente matrice. Il codice sorgente è identico a quello dell'esercizio precedente.

```
A:
    15    -2     2
     1    10    -3
    -2     1     0

spettro:
  5.1208e-001  1.4103e+001  1.0385e+001

autovettori corrispondenti:
 -8.8117e-002 -9.4359e-001  3.9293e-001
  3.0874e-001 -3.1169e-001  9.1948e-001
  9.4706e-001  1.1172e-001  1.2866e-002

METODO DELLE POTENZE
autovalore di modulo max [#iterazioni: 43]: 1.410255e+001
ratioeig: 0.736417
autovettore corrispondente:
    9.435920410960773e-001
```

```
    3.116945006717176e-001
   -1.117165978319585e-001

METODO DELLE POTENZE CON SPOSTAMENTO [shift: 2.500000e+000]
autovalore [#iterazioni: 10]: 5.120856e-001
ratioeig: 0.252102
autovettore corrispondente:
   -8.811711505090356e-002
    3.087389413811202e-001
    9.470563025026396e-001

METODO DELLE POTENZE CON SPOSTAMENTO [shift: 1.500000e+000]
autovalore [#iterazioni: 8]: 5.120848e-001
ratioeig: 0.111185
autovettore corrispondente:
   -8.811725638582039e-002
    3.087386839307303e-001
    9.470563732807863e-001

METODO DELLE POTENZE INVERSE
autovalore di modulo min [#iterazioni: 6]: 5.120848e-001
ratioeig: 0.049308
autovettore corrispondente:
   -8.811725678187253e-002
    3.087386819609979e-001
    9.470563738860656e-001
```

11.7 Cerchi di Gerschgorin

```
function [ ] = circGerschgorin(A)
% CIRCGERSCHGORIN disegna i cerchi di Gerschgorin
n = max(size(A));
figure;
x = linspace(0,2*pi);
for k = 1:n
    C = A(k,k);
    if(isreal(A(k,k)))
        C = complex(C,0);
    end
    r = sum(A(k,1:k-1))+sum(A(k,k+1:n));
    circ = r*exp(x.*i)+C;
    plot(circ)
    hold on
    plot(C,'r.','LineWidth',1)
    hold on
```

11.8 Parametro di shift

Studiare il metodo delle potenze con spostamento al variare del parametro di shift per la seguente matrice

$$A = \begin{bmatrix} 1 & i & 3 & 4 \\ -i & 4 & 5-i & i \\ i & 2i & 7i & 0 \\ 2+2i & 1 & 1 & -2i \end{bmatrix}$$

Nota Il grafico `convergenza agli autovalori di A` associa ad ogni parametro di shift μ l'autovalore cui il metodo delle potenze con spostamento μ converge, secondo la seguente convenzione:

10. $-2.857912109054695 - 2.570434616912589i$
20. $1.745873352909238 + 6.043623984227994i$
30. $3.688656836040702 + 0.318094056860421i$
40. $2.423381920104756 + 1.208716575824174i$

```
A=[1,i,3,4;-i,4,5-i,i;i,2i,7i,0;2+2i,1,1,-2i]
circGerschgorin(A)
E = eig(A);
plot(real(E),imag(E),'g+')
axis([-7 7 -7 7])
axis equal
figure
plot(E,'MarkerSize',20,'Marker','+','LineWidth',2,'LineStyle',...
     'none','Color',[1 1 1]);
X = [-7:0.05:7];
Y = [-7:0.05:7];
Z = zeros(max(size(X)),max(size(Y)));
W = Z;
Maxit =200;
tol = 10^(-15);
for k = 1: max(size(X))
    for j = 1: max(size(Y))
        y = ones(max(size(A)),1);
        [y,lambda,num] = pot_inv2(A,y,Maxit,tol,X(k)+i*Y(j));
        Z(k,j) = num;
        for h = 1 : max(size(A))
            if(abs(E(h)-lambda)<10^(-6))
                W(k,j) = 10*h;
            end
        end
    end
end
hold on
```

```
contourf(X,Y,Z','LineStyle','none','LineColor',[0 0 0],'Fill','on')
axis([-7 7 -7 7])
axis equal
%
figure
contourf(X,Y,W','LineStyle','none','LineColor',[0 0 0],'Fill','on')
hold on
plot(real(E),imag(E),'MarkerSize',20,'Marker','+','LineWidth',2,...
     'LineStyle','none','Color',[1 1 1]);
axis([-7 7 -7 7])
axis equal
```

Osservazioni

La convergenza del metodo aumenta in prossimità dell'autovalore. Il luogo dei punti ove il metodo non converge in almeno 200 iterazioni stima il luogo dei punti per cui la matrice $(A - \mu I)$ ammette autovalori minimi di modulo uguale, condizione sotto la quale il metodo delle potenze inverse non converge.

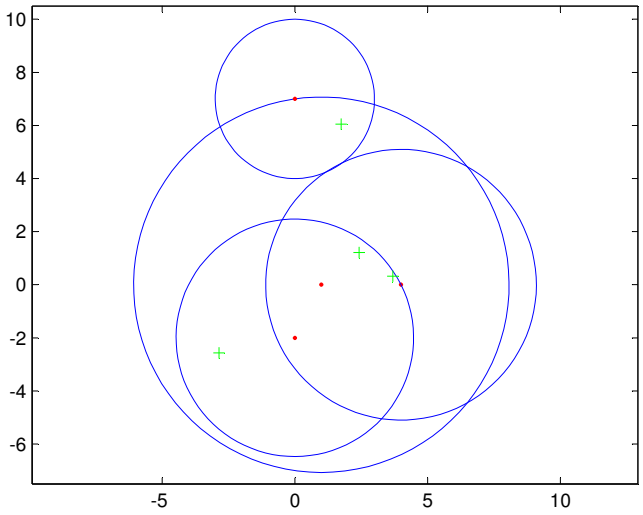

Figura 24: cerchi di Gerschgorin della matrice a coefficienti complessi `A = [1,i,3,4;-i,4,5-i,i;i,2i,7i,0;2+2i,1,1,-2i]`

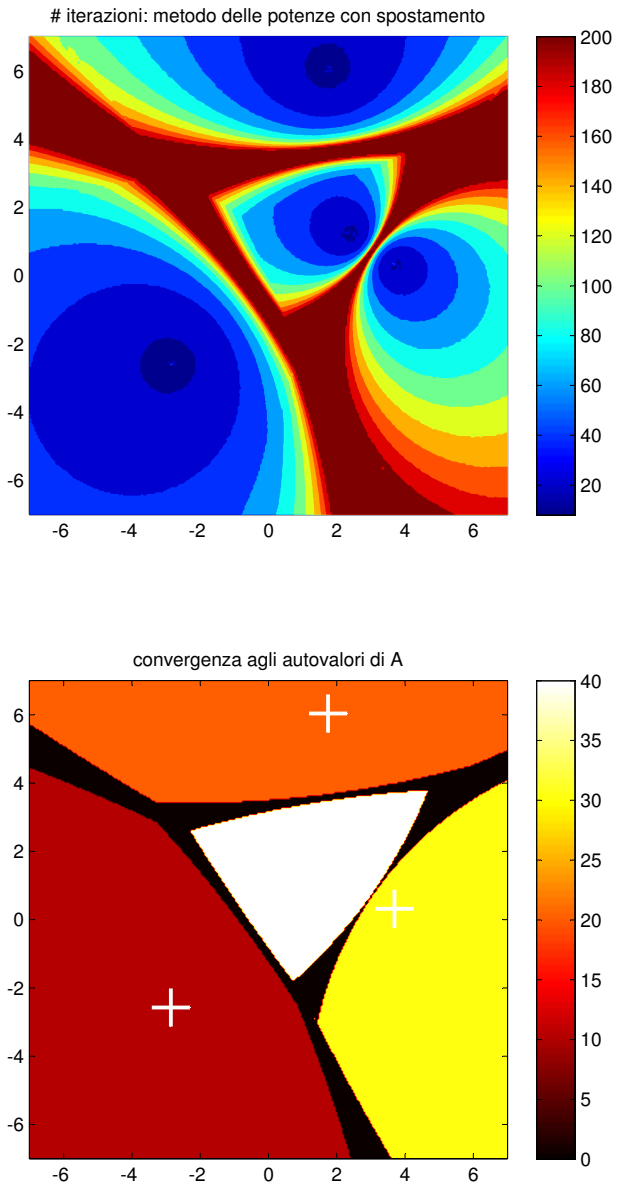

Figura 25: applicazione del metodo delle potenze con spostamento al variare del parametro di shift: per ogni parametro visualizziamo numero di iterazioni del metodo e autovalore cui converge.

Parte IV
EQUAZIONI NON LINEARI

12 ESERCITAZIONE 9.1

12.1 Metodo di bisezione

```
function [x,it] = bisez(fname,a,b,tol)
%BISEZ metodo di bisezione per il calcolo degli zeri di una funzione
% Dati input:
%  fname    STRING title function della funzione f
%  [a b]    DOUBLE estremi di bracket
%  tol      DOUBLE tolleranza
%
% Dati output:
%  x        DOUBLE zero della funzione f
%  it       DOUBLE # iterazioni
    maxit = round(log((b-a)/tol)/log(2)+1);
    fa = feval(fname,a);
    fb = feval(fname,b);
    if sign(fa)*sign(fb)>=0
        error('intervallo non corretto')
    else
    it = 0;
    while(abs(b-a)>=tol+eps(max(abs(a),abs(b)))&&it<=maxit)
        it = it+1;
        pm = a+(b-a)/2;
        fpm = feval(fname,pm);
        if(fpm == 0)
            break;
        end
        if(sign(fpm)*sign(fa)>0)
            a=pm;
            fa=fpm;
        else
            b=pm;
            fb=fpm;
        end
    end
    if(it>maxit)
        disp('superato il numero massimo di iterazioni')
    else
        x = pm;
    end
```

12.2 Metodo di Newton

```
function [x,it] = nexton(fname, dfname,xo, tolx, tolf, maxit)
%NEWTON metodo di newton per il calcolo degli zeri di una funzione
% Dati input:
%  fname    STRING title function della funzione f
%  dfname   STRING title funcion della derivata della funzione f
%  tolx     DOUBLE tolleranza (criterio d'arresto dell'incremento)
%  tolf     DOUBLE tolleranza (criterio d'arresto del residuo)
%  maxit    DOUBLE # massimo iterazioni
%
% Dati output:
%  x        DOUBLE zero della funzione f
%  it       DOUBLE # iterazioni

    x = xo;
    fx = feval(fname,x);
    fprintf('it        x             f(x)\n')
    for i=1:maxit
        d = fx/feval(dfname,x);
        x = x-d;
        fx = feval(fname,x);
        fprintf('%g     %d     %d\n',i,x,fx)
        if(abs(fx)<tolf && abs(d)<tolx || fx==0)
            break;
        end
    end
    it = i;
    if(it>maxit)
        disp('superato il numero massimo di iterazioni')
    end
end
```

Newton modificato: radici multiple (`newton_m`)

```
function [x,it] = nexton_m(fname, dfname,xo, tolx, tolf, maxit,s)
%NEWTON_M metodo di newton per il calcolo degli zeri di una funzione con
%         molteplicita' s.
% ...
%  s        DOUBLE coefficiente correttivo (molteplicita' della radice)
    for ...
     d = s*fx/feval(dfname,x);
    ...
```

Newton modificato: plot (newton_grafico)

```
    if(d<0)
        y = linspace(x,x+d,2);
    else
        y = linspace(x+d,x,2);
    end
    f = c*(y-x);
    hold on
    plot(y,f,'r','LineWidth',1)

    w = [0:fx/2:fx];
    z = x*ones(1,max(size(w)));
    hold on
    plot(z,w,'r','LineWidth',1)
```

12.3 Metodo delle secanti

```
function [x,it] = secanti(fname,x0,x1,tolx,tolf,maxit)
%SECANTI metodo delle secanti per il calcolo degli zeri di una funzione
% Dati input:
%  fname    STRING title function della funzione f
%  dfname   STRING title funcion della derivata della funzione f
%  tolx     DOUBLE tolleranza (criterio d'arresto dell'incremento)
%  tolf     DOUBLE tolleranza (criterio d'arresto del residuo)
%  maxit    DOUBLE # massimo iterazioni
%
% Dati output:
%  x        DOUBLE zero della funzione f
%  it       DOUBLE # iterazioni

    fprintf('it        x\n')
    for i=1:maxit
        fx0 = feval(fname,x0);
        fx1 = feval(fname,x1);
        if(fx0 == fx1)
            error('method secanti fails: fxo == fx1')
        end
        d = (x1-x0)/(fx1-fx0)*fx1;
        x = x1-d;
        fx = feval(fname,x);
        fprintf('%g      %g\n',i,x)
        if(abs(fx)<tolf && abs(d)<tolx || fx==0)
            break;
        else
            x0 = x1;
```

```
            x1 = x;
        end
    end
    it=i;
    if(it>maxit)
        disp('superato il numero massimo di iterazioni')
    end
end
```

12.4 *esercizio 1*

Assegnato il seguente polinomio

$$\begin{aligned} f(x) &= (x-1)(x-2)(x-3)(x-4)(x-5)(x-6)(x-7) \\ &= x^7 - 28x^6 + 322x^5 - 1960x^4 + 6769x^3 - 13132x^2 + 13068x - 5040 \end{aligned}$$

determinare le radici di $f_\epsilon(x) = f(x) + \epsilon x^6$ con $\epsilon = 0.002$.

```
a = [1,-28,322,-1960,6769,-13132,13068,-5040]'
e = 0.002;
a_mod = [1,-28+e,322,-1960,6769,-13132,13068,-5040]'

r1 = roots(a);
r2 = roots(a_mod);
fprintf('RADICI DEL POLINOMIO DI COEFFICIENTI a:\n')
disp(r1)
fprintf('RADICI DEL POLINOMIO DI COEFFICIENTI a_mod:\n')
disp(r2)

err_e = e/a(1);
err = zeros(max(size(r1)),1);
for i = 1: max(size(err))
    err(i) = abs(r1(i)-r2(i))/abs(r1(i));
end
fprintf('ERRORE COEFFICIENTE I: %d\n',err_e)
fprintf('ERRORE RADICI DEL POLINOMIO PERTURBATO:\n')
disp(err)

figure;
plot(r1','ro')
plot(r1,zeros(max(size(r1)),1),'ro')
hold on
plot(r2,'o')
legend('radici di p(x)','radici di p(x) perturbato')
```

```
a =        1
         -28
         322
       -1960
        6769
      -13132
       13068
       -5040

a_mod =
    1.000000000000000e+000
   -2.799800000000000e+001
    3.220000000000000e+002
   -1.960000000000000e+003
    6.769000000000000e+003
   -1.313200000000000e+004
    1.306800000000000e+004
   -5.040000000000000e+003

RADICI DEL POLINOMIO DI COEFFICIENTI a:
    7.000000000000703e+000
    5.999999999997170e+000
    5.000000000004296e+000
    3.999999999996843e+000
    3.000000000001224e+000
    1.999999999999756e+000
    1.000000000000016e+000

RADICI DEL POLINOMIO DI COEFFICIENTI a_mod:
     6.644428959584060e+000 +3.907692289827950e-001i
     6.644428959584060e+000 -3.907692289827950e-001i
     4.368166209320691e+000 +2.244381249933771e-001i
     4.368166209320691e+000 -2.244381249933771e-001i
     2.971740865485022e+000
     2.001071574418111e+000
     9.999972222873942e-001

ERRORE COEFFICIENTE I: 2.000000e-003
ERRORE RADICI DEL POLINOMIO PERTURBATO:
    7.547554747997910e-002
    1.256084385738502e-001
    1.341024102690752e-001
    1.077957645384108e-001
    9.419711505396966e-003
    5.357872091775282e-004
    2.777712621360130e-006
```

Osservazioni

Il calcolo delle radici del polinomio $\prod_{k=1}^{n}(x-k)$ è malcondizionato. Sia f(x) un polinomio a coefficienti in $\mathbb{R}$ (o in $\mathbb{C}$)

$$f(x) = x^n + a_1 x^{n-1} + \dots a_{n-1}x + a_n$$

Si definisca $r = r(a_1, \dots, a_n)$ la funzione che ad ogni n-upla di coefficienti $(a_1, \dots, a_n)$ associa una radice semplice dell'equazione $f(x) = 0$.

$$f(r) = r^n(a_1, \dots, a_n) + a_1 r^{n-1}(a_1, \dots, a_n) + \dots a_{n-1} r(a_1, \dots, a_n) + a_n \equiv 0$$

$$\frac{\partial f}{\partial a_k}(a_1, \dots, a_n, r(a_1, \dots, a_n)) = f'(r)\frac{\partial r}{\partial a_k} + r^{n-k} \equiv 0$$

Ricordando che la derivata stima il rapporto incrementale, l'errore relativo sulla radice r provocata da una perturbazione sul coefficiente a_k

$$\frac{\Delta r}{r} \sim -a_k \frac{r^{n-k-1}}{f'(r)} \frac{\Delta a_k}{a_k}$$

(in generale, essendo $r : \mathbb{R}^n \longrightarrow \mathbb{R}$, $\frac{\mathbf{\Delta} r}{r} = \frac{\sqrt{\Delta_1 r^2 + \dots + \Delta_n r^2}}{r}$, ove $\frac{\Delta_i r}{r}$ è la perturbazione sul coefficiente i-esimo).
Nel calcolo della radice $r = 6$,

$$\frac{\Delta r}{r} \sim 28 \frac{6^4}{5!} \frac{\Delta a_k}{a_k} \sim 0.0216$$

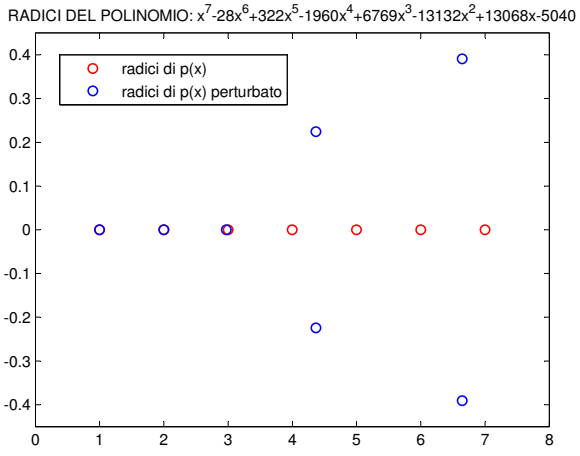

Figura 26: radici del polinomio $p(x)$ e $p^*(x) = p(x) + \epsilon x^6$

12.5 *esercizio 2*

Confrontare i metodi di bisezione, Newton e secanti per le seguenti funzioni:

fun_2_1 $f(x) = \sin x - \frac{x^2}{4}$

fun_2_2 $f(x) = x^6 - x - 1$

fun_2_3 $f(x) = e^{-x} + \sin x$

```
[x,it] = bisez('fun_2_1',1,2,1e-6);
fprintf('METODO DI BISEZIONE [# iter: %g]:\n',it)
fprintf('x: %d\n',x)
fprintf('\nMETODO DI NEWTON:\n')
[x,it] = nexton('fun_2_1','dfun_2_1',2,1e-6,1e-8,500);
fprintf('it: %d\n',it)
fprintf('x: %d\n',x)
fprintf('\nMETODO DELLE SECANTI:\n')
[x,it] = secanti('fun_2_1',1,2,1e-6,1e-8,500);
fprintf('it: %d\n',it)
fprintf('x: %d\n',x)
fprintf('-----------------------------------------------------\n')

[x,it] = bisez('fun_2_2',1,2,1e-4);
fprintf('METODO DI BISEZIONE [# iter: %g]:\n',it)
fprintf('x: %d\n',x)
fprintf('\nMETODO DI NEWTON:\n')
[x,it] = nexton('fun_2_2','dfun_2_2',2,1e-4,1e-6,500);
fprintf('it: %d\n',it)
fprintf('x: %d\n',x)
fprintf('\nMETODO DELLE SECANTI:\n')
[x,it] = secanti('fun_2_2',1,2,1e-4,1e-6,500);
fprintf('it: %d\n',it)
fprintf('x: %d\n',x)
fprintf('-----------------------------------------------------\n')

[x,it] = bisez('fun_2_3',0,8,1e-6);
fprintf('METODO DI BISEZIONE [# iter: %g]:\n',it)
fprintf('x: %d\n',x)
fprintf('\nMETODO DI NEWTON:\n')
[x,it] = nexton('fun_2_3','dfun_2_3',0,1e-6,1e-8,500);
fprintf('it: %d\n',it)
fprintf('x: %d\n',x)
fprintf('\nMETODO DELLE SECANTI:\n')
[x,it] = secanti('fun_2_3',0,1,1e-6,1e-8,500);
fprintf('it: %d\n',it)
fprintf('x: %d\n',x)
```

```
METODO DI BISEZIONE [# iter: 20]:
x: 1.933753e+000

METODO DI NEWTON:
it        x             f(x)
1     1.935951e+000     -2.908231e-003
2     1.933756e+000     -3.454612e-006
3     1.933754e+000     -4.899636e-012
4     1.933754e+000     -2.220446e-016
it: 4
x: 1.933754e+000

METODO DELLE SECANTI:
it        x
1      1.86704
2      1.93135
3      1.93384
4      1.93375
5      1.93375
it: 5
x: 1.933754e+000
-------------------------------------------------------
METODO DI BISEZIONE [# iter: 14]:
x: 1.134705e+000

METODO DI NEWTON:
it        x             f(x)
1     1.680628e+000     1.985294e+001
2     1.430739e+000     6.146795e+000
3     1.254971e+000     1.651657e+000
4     1.161538e+000     2.943097e-001
5     1.136353e+000     1.682607e-002
6     1.134731e+000     6.573837e-005
7     1.134724e+000     1.015406e-009
it: 7
x: 1.134724e+000

METODO DELLE SECANTI:
it        x
1      1.01613
2      1.03067
3      1.17569
4      1.12368
5      1.13367
6      1.13475
7      1.13472
```

```
it: 7
x: 1.134724e+000
-----------------------------------------------------
METODO DI BISEZIONE [# iter: 23]:
x: 6.285048e+000

METODO DI NEWTON:
it        x             f(x)
1     5.000000e-001     1.271051e-001
2     5.856438e-001     4.011277e-003
3     5.885294e-001     4.620254e-006
4     5.885327e-001     6.160961e-012
5     5.885327e-001     0
it: 5
x: 5.885327e-001

METODO DELLE SECANTI:
it        x
1      0.678614
2      0.569062
3      0.58926
4      0.588538
5      0.588533
6      0.588533
it: 6
x: 5.885327e-001
```

Osservazioni

Il metodo di Newton converge generalmente più velocemente del metodo delle secanti, che converge più velocemente del metodo di bisezione.
La convergenza del metodo dipende dalla scelta dei parametri di innesco. Il calcolo della radice positiva di modulo minimo richiede una conoscenza aprioristica della bracket d'innesco (attraverso stime grafiche o analitiche: calcolo della monotonia della funzione, dei suoi massimi e del loro segno). A patto di scegliere un intervallo che sia effettivamente una bracket ($[a, b]$ *t.c.* $f(a)f(b) < 0$), il metodo converge ad uno zero della radice; non necessariamente il minimo positivo.

```
[x,it] = bisez('fun_2_3',0,8,1e-6);

METODO DI BISEZIONE [# iter: 23]:
x: 6.285048e+000
-----------------------------------------------------
[x,it] = bisez('fun_2_3',0,1,1e-6);
```

```
METODO DI BISEZIONE [# iter: 20]:
x: 5.885324e-001
-------------------------------------------------------
[x,it] = bisez('fun_2_3',-10,10,1e-6);

Error using bisez (line 15)
intervallo non corretto

Error in esercizio2 (line 30)
[x,it] = bisez('fun_2_3',-10,10,1e-6);
```

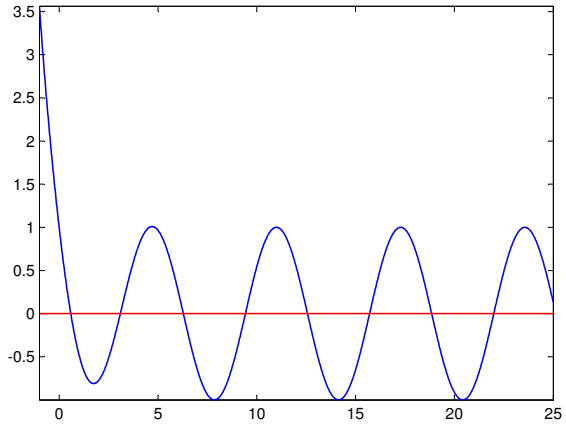

Figura 27: $f(x) = e^{-x} + \sin x$

12.6 *esercizio 3*

Determinare con il metodo di Newton le radici di $f(x) = \arctan(x)$.

```
fprintf('METODO DI NEWTON:\n')
[x,it] = nexton('fun_3','dfun_3',1.4,1e-5,1e-6,20);
fprintf('\nit: %d\n',it)
fprintf('x: %d\n',x)

figure;
x=linspace(-10,10);
f=inline('atan(x)');
```

```
y=f(x);
plot(x,y);
[x,it] = nexton_grafico('fun_3','dfun_3',1.4,1e-5,1e-6,7);
```

```
METODO DI NEWTON:
it          x                f(x)
1     -1.413619e+000     -9.551183e-001
2      1.450129e+000      9.670887e-001
3     -1.550626e+000     -9.980141e-001
4      1.847054e+000      1.074578e+000
5     -2.893562e+000     -1.238051e+000
6      8.710326e+000      1.456491e+000
7     -1.032498e+002     -1.561111e+000
8      1.654056e+004      1.570736e+000
9     -4.297215e+008     -1.570796e+000
10     2.9006411+017      1.570796e+000
11    -1.321624e+035     -1.570796e+000
12     2.743694e+070      1.570796e+000
13    -1.182473e+141     -1.570796e+000
14     2.196354e+282      1.570796e+000
15    -Inf               -1.570796e+000
16     NaN                NaN
17     NaN                NaN
18     NaN                NaN
19     NaN                NaN
20     NaN                NaN

it: 20
x: NaN
```

Osservazioni

Il metodo di Newton non converge per nessun dato fuori da un opportuno intorno di zero.

$$x_1 = x_0 - \arctan(x_0)(1 + x_0^2)$$

Mostriamo che $x_1 x_0 < 0$

$$f(x) = x - \arctan(x)(1 + x^2)$$

$$f'(x) = -2x \arctan(x) < 0 \quad \forall x \in \mathbb{R} \setminus \{0\}$$

Pertanto f è monotona decrescente e poiché $f(0) = 0$, segue la tesi. Mostriamo che $|x_1| < |x_0|$ in $\mathcal{U}(0)$ e $|x_1| > |x_0|$ altrimenti

$$|x_1| - |x_0| = \arctan(|x|)(1 + x^2) - 2|x|$$

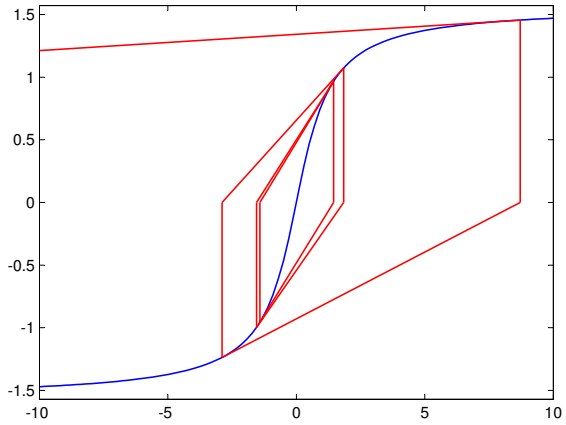

Figura 28: non convergenza del metodo di Newton: $f(x) = \arctan(x)$

$$\arctan(|x|)(1+x^2) - 2|x| \sim -x + \frac{2}{3}x^3 + o(x^3) \qquad x \to 0^+$$

$$\arctan(|x|)(1+x^2) - 2|x| \sim \frac{\pi}{2}x^2 \qquad x \to +\infty$$

12.7 *esercizio 4*

Determinare con il metodo di Newton le radici di $p(x) = (x-1)^{10}$.

```
fprintf('METODO DI NEWTON:\n')
[x,it] = nexton('fun_4','dfun_4',2,1e-5,1e-6,500);
fprintf('\nit: %d\n',it)
fprintf('x:')
disp(x)

figure;
x=linspace(0.1,2.1);
f=inline('(x-1).^10');
y=f(x);
plot(x,y);
axis([0.1 2.1 -0.1 1.1]);
[x,it] = nexton_grafico('fun_4','dfun_4',2,1e-5,1e-6,10);


fprintf('METODO DI NEWTON MODIFICATO [coeff correttivo: 10]:\n')
```

```
[x,it] = nexton_m('fun_4','dfun_4',2,1e-5,1e-6,500,10);
fprintf('\nit: %d\n',it)
fprintf('x:')
disp(x)

a = 0.5;
b = 20;
x = [a:0.1:b];
n = max(size(x));
iter = zeros(1,n);
count=1;
for i=a:0.1:b
    [w,it] = nexton_m('fun_4','dfun_4',2,1e-5,1e-6,100,i);
    iter(count) = it;
    count=count+1;
end
figure;
plot(x,iter,'LineWidth',1)
axis([a b -0.1 100.1]);
xlabel('coefficiente correttivo');
ylabel('# iterazioni');
```

```
METODO DI NEWTON:
it          x               f(x)
1     1.900000e+000     3.486784e-001
2     1.810000e+000     1.215767e-001
...
88     1.000094e+000     5.412630e-041
89     1.000085e+000     1.887267e-041

it: 89
x:    1.000084641497829e+000

METODO DI NEWTON MODIFICATO [innesco: 2; coeff correttivo: 10]:
it          x               f(x)
1           1                0
it: 1
x:  1

METODO DI NEWTON MODIFICATO [innesco: 2.1; coeff correttivo: 10]:
it        x             f(x)
1     1.000000e+000     2.913414e-157
2     1     0

it: 2
x:  1
```

Osservazioni

Il metodo di Newton converge linearmente per radici di molteplicità multipla. Sia ξ radice semplice di $f(x)$, i.e. $f(\xi) = 0$ e $f'(\xi) \neq 0$,

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = g(x)$$

$$g'(x) = \frac{f(x)f''(x)}{[f'(x)]^2} \qquad g'(\xi) = 0$$

$$g''(x) = \frac{(f'(x)f''(x) + f(x)f'''(x))[f'(x)]^2 - 2f(x)f'(x)[f''(x)]^2}{[f'(x)]^4} \quad g''(\xi) = \frac{f''(\xi)}{f'(\xi)}$$

Se $f''(\xi) \neq 0$ (da notare che $f''(0) = 0$ per $f(x) = \arctan x$),

$$\lim_{x \to +\infty} \frac{e_{n+1}}{[e_n]^2} \neq 0$$

Pertanto, se converge, il metodo di Newton ha ordine $p = 2$. Sia $\mu > 1$ la molteplicità della radice ξ, i.e. $f(\xi) = f'(\xi) = \ldots = f^{(\mu-1)}(\xi) = 0$ e $f^{(\mu)}(\xi) \neq 0$, $e_n = \xi - x_n$ e $\eta_n, \delta_n \in (\xi, x_n)$ o (x_n, ξ)

$$e_{n+1} = e_n - \frac{\frac{e_n^\mu}{\mu!} f^{(\mu)}(\eta_n)}{\frac{e_n^{\mu-1}}{(\mu-1)!} f^{(\mu)}(\delta_n)} = e_n \left[1 - \frac{1}{\mu} \frac{f^{(\mu)}(\eta_n)}{f^{(\mu)}(\delta_n)}\right]$$

$$\lim_{x \to +\infty} \frac{e_{n+1}}{e_n} = \frac{1-\mu}{\mu}$$

L'ordine di convergenza scende a $p = 1$. Per ripristinare la convergenza quadratica è sufficiente introdurre un coefficiente correttivo μ (in figura si osservi la variazione della velocità di convergenza in funzione di μ: il minimo corrisponde alla molteplicità della radice del polinomio):

$$x_{n+1} = x_n - \mu \frac{f(x_n)}{f'(x_n)}$$

Alternativamente, qualora non fosse nota la molteplicità della radice, è possibile applicare il metodo di Newton alla funzione $g(x) = \frac{f(x)}{f'(x)}$, t.c. $g(\xi) = 0$ e $g'(\xi) = 1 - \frac{1}{\mu}$ (dim. per reiterazione della regola di de l'Hôpital)

$$x_{n+1} = x_n - \frac{f(x_n)f'(x_n)}{[f'(x_n)]^2 - f(x_n)f''(x_n)}$$

La convergenza è nuovamente quadratica, ma si richiede il calcolo di $f''(x)$. Nel caso di specie $g(x)$ è banalmente un'applicazione lineare.

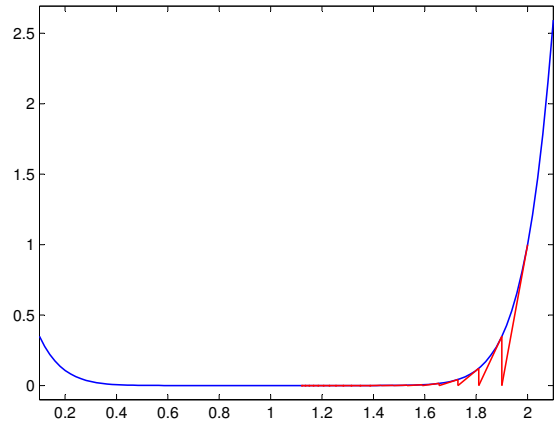

Figura 29: convergenza lineare del metodo di Newton: $p(x) = (x-1)^{10}$

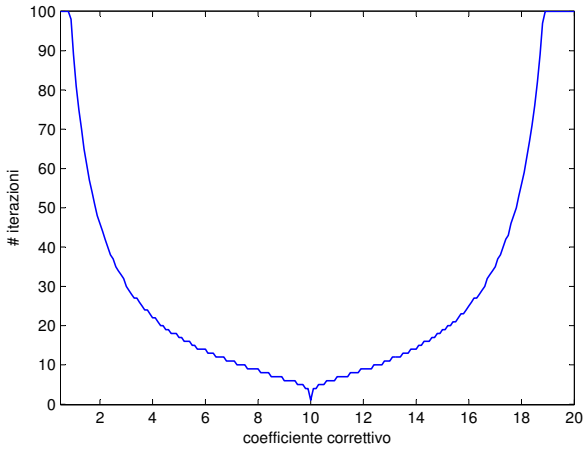

Figura 30: ripristino della convergenza quadratica del metodo di Newton: `newton_m`. Il minimo numero di iterazioni del metodo corrisponde alla molteplicità della radice del polinomio

12.8 *esercizio 5*

Determinare con il metodo di Newton modificato le radici di

$$p(x) = x^3 + x^2 - 33x + 63$$

```
fprintf('METODO DI NEWTON:\n')
[x,it] = nexton('fun_5','dfun_5',2,1e-5,1e-6,100);
fprintf('\nit: %d\n',it)
fprintf('x:')
disp(x)

fprintf('METODO DI NEWTON MODIFICATO [coeff correttivo: 2]:\n')
[x,it] = nexton_m('fun_5','dfun_5',2,1e-5,1e-6,100,2);
fprintf('\nit: %d\n',it)
fprintf('x:')
disp(x)

[x,fval,exitflag,output] = fzero('fun_5',5)
```

```
METODO DI NEWTON:
it           x              f(x)
1      2.529412e+000     2.110320e+000
2      2.770663e+000     5.138939e-001
3      2.886693e+000     1.269299e-001
4      2.943673e+000     3.154854e-002
5      2.971917e+000     7.864670e-003
6      2.985978e+000     1.963391e-003
7      2.992994e+000     4.905026e-004
8      2.996498e+000     1.225826e-004
9      2.998249e+000     3.064029e-005
10     2.999125e+000     7.659402e-006
11     2.999562e+000     1.914767e-006
12     2.999781e+000     4.786812e-007
13     2.999891e+000     1.196690e-007
14     2.999945e+000     2.991709e-008
15     2.999973e+000     7.479244e-009
16     2.999986e+000     1.869815e-009
17     2.999993e+000     4.674519e-010

it: 17
x:    2.999993162945802e+000

METODO DI NEWTON MODIFICATO [coeff correttivo: 2]:
it           x              f(x)
1      3.058824e+000     3.480562e-002
```

```
2      3.000171e+000     2.941178e-007
3      3.000000e+000     7.105427e-015
4      3.000000e+000     2.330580e-012

it: 4
x:    2.999999518028236e+000

-------------------------------------------------
x = -7
fval = 0
exitflag = 1
output =
    intervaliterations: 14
            iterations: 6
             funcCount: 34
             algorithm: 'bisection, interpolation'
               message: 'Zero found in the interval [-7.8, 14.051]'
```

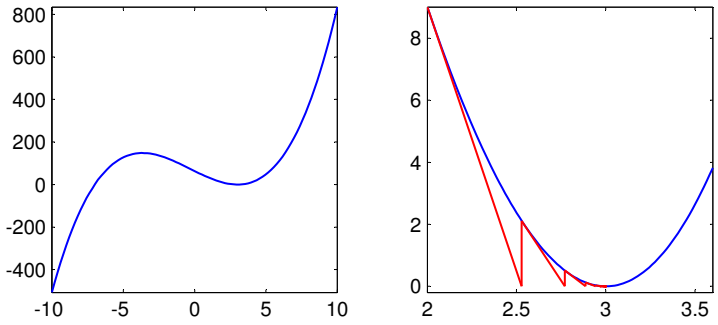

Figura 31: metodo di Newton modificato: $f(x) = x^3 + x^2 - 33x + 63$

Osservazioni

La function `fzero` non riesce a determinare la presenza della radice doppia $\xi = 3$. Salvo tentare esattamente la radice, la ricerca degli estremi di bracket attorno alla radice doppia fallisce: $f(x) > 0 \quad \forall x \in \mathcal{U}(3)$, opportuno intorno di 3.
Poiché il parametro d'innesco non può essere determinato con qualche iterazione del metodo di bisezione, analiticamente

$$\lim_{x \to -\infty} p(x) = -\infty \qquad \lim_{x \to +\infty} p(x) = +\infty$$

$$p'(x) = 3x^2 + 2x - 33$$

Per la regola di Cartesio (da notare che $\Delta_{p'(x)} > 0$) ammette due radici reali di segno discorde e $p(0) = 63$; $p(x)$ ammette un solo massimo per $x < 0$ e un solo minimo per $x > 0$. Pertanto, $p(x)$ ammette una sola radice reale negativa e due radici reali positive (coincidenti o meno) o due radici complesse.

$$p''(x) = 6x + 2x$$

l'unico flesso è raggiunto in $x = \frac{1}{3}$; per $x > 0$ la funzione è convessa. È ragionevole credere che per ogni parametro d'innesco positivo il metodo converga alle radici positive.

12.9 *esercizio 6*

Determinare la più piccola radice positiva di $f(x) = \tan(x) - x$.
Gli estremi di bracket si scelgano tra $\left[\frac{\pi}{2}, \frac{3}{2}\pi\right]$. Nell'intervallo $[0, \frac{\pi}{2}]$, $f(x) = 0$ per $x = 0$ e la funzione è convessa, i.e. (nell'intervallo) la retta $y = x$, tangente al grafico di $\tan(x)$ in 0, non lo interseca in altri punti. La prima intersezione positiva appartiene all'intervallo $\left[\frac{\pi}{2}, \frac{3}{2}\pi\right]$, giacché il suo codominio è $\mathbb{R}$.

```
[innesco,itb] = bisez('fun_6',pi/2+0.1,3/2*pi-0.1,1e-1);
fprintf('METODO DI BISEZIONE [# iter: %g]:\n',itb)
fprintf('innesco: %d\n',innesco)

fprintf('METODO DI NEWTON:\n')
[xn,itn] = nexton('fun_6','dfun_6',innesco,1e-8,1e-8,100);
fprintf('\nit_newton: %d\n',itn)
fprintf('it_tot = it_bisec + it_newton: %d\n',itb+itn)
fprintf('x:')
disp(xn)

fprintf('METODO DELLE SECANTI:\n')
[xs,its] = secanti('fun_6',innesco,innesco+0.1,1e-8,1e-8,100);
fprintf('it_secanti: %d\n',its)
fprintf('it_tot = it_bisec + it_secanti: %d\n',itb+its)
fprintf('x: %d\n',xs)
```

```
METODO DI BISEZIONE [# iter: 5]:
innesco: 4.520464e+000

METODO DI NEWTON:
it          x              f(x)
1     4.496836e+000     7.030985e-002
2     4.493465e+000     1.116823e-003
3     4.493409e+000     2.911742e-007
4     4.493409e+000     1.865175e-014
5     4.493409e+000     8.881784e-016
```

```
it_newton: 5
it_tot = it_bisec + it_newton: 10
x:    4.493409457909064e+000

METODO DELLE SECANTI:
it        x
1      4.50929
2      4.50274
3      4.49411
4      4.49344
5      4.49341
6      4.49341
7      4.49341
it_secanti: 7
it_tot = it_bisec + it_secanti: 12
x: 4.493409e+000
```

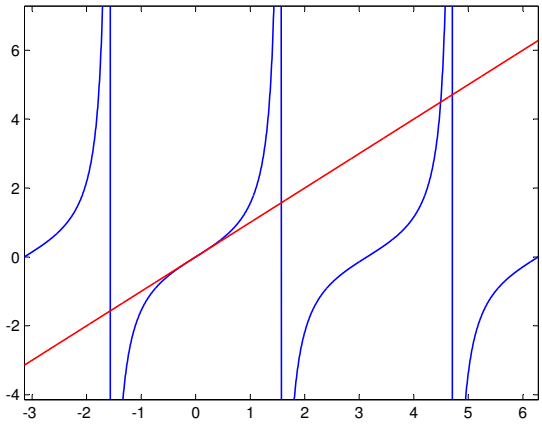

Figura 32: $\tan(x) - x = 0$

12.10 *esercizio 7*

Convertire l'equazione $x^2 - 5 = 0$ al problema di punto fisso

$$x = x + c(x^2 - 5) \equiv g_c(x)$$

Si scelga il valore di c che assicuri la convergenza di $x_{n+1} = x_n + c(x_n^2 - 5)$ a $\xi = \sqrt{5}$. Verificare il tutto applicando il metodo di iterazione del punto fisso.

Scelgo c in modo che $g'(\xi) \sim 0$ (la condizione sufficiente di convergenza richiede $g'(\xi) < 1$; perché il test d'arresto dell'incremento sia efficiente si richiede $g'(\xi) \sim 0$). In aritmetica esatta, $g'(x) = 1 + 2c\xi = 0 \Rightarrow c = -\frac{1}{2\sqrt{5}}$.
Scelgo dunque $c = -\frac{1}{4}$,

$$g_{-\frac{1}{4}}(x) = x - \frac{1}{4}(x^2 - 5)$$

$$\left|g'_{-\frac{1}{4}}(x)\right| = \left|1 - \frac{1}{2}x\right| < 1 \quad \forall x \in (0, 4)$$

Poiché $g((0,4)) = (\frac{5}{4}, \frac{9}{4})$, la mappa $g_{-\frac{1}{4}} : [\epsilon, 4-\epsilon] \to [\epsilon, 4-\epsilon]$ è una contrazione. Fissato c come sopra, per un qualsiasi dato iniziale in $[\epsilon, 4-\epsilon]$ il metodo di punto fisso converge.

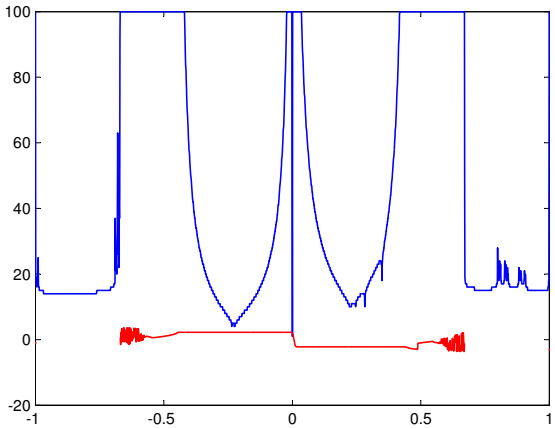

Figura 33: # iterazioni del metodo di punto fisso al variare del coefficiente di convergenza (in blu); valore cui il metodo converge (i.e. valore all'iterata massima) al variare del coefficiente di convergenza (in rosso)

```
tol = 10^(-6);
maxit = 100;
c = -1/4;
x = 2;
x0 = 1;
it = 0;
while abs(x-x0)>=tol && it< maxit
    it = it+1;
```

```
    x0 = x;
    x = x+c*(x^2-5);
end

fprintf('METODO DI PUNTO FISSO [# iter: %d]:\n'
  ...'pto valutato: %1.16f \n pto fisso:'
  ...'%1.16f\n',it,x,sqrt(5))
```

```
METODO DI PUNTO FISSO [# iter: 7]:
 pto valutato: 2.2360674917056476
 pto fisso:    2.2360679774997898
```

12.11 *esercizio 8*

Approssimare la soluzione di $x = \sqrt{x+1}$ con il metodo di iterazione di punto fisso. Sia $\phi : [0, +\infty) \to [0, +\infty)$ t.c. $\phi(x) = \sqrt{x+1}$, la mappa ϕ è una contrazione: $|\phi'(x)| < \frac{1}{2}$.
Procedendo come nell'esercizio precedente è possibile accelerare la convergenza, modificando la funzione ϕ in $\phi^*(x) = x + c(x - \sqrt{x+1})$ (ϕ e ϕ^* hanno evidentemente lo stesso punto fisso). Scelto $c = -\frac{3}{2}$ e provato che su $\mathbb{R}^+$ ϕ^* è ancora una contrazione, il metodo di punto fisso converge velocemente:

```
METODO DI PUNTO FISSO [c: -1.5; xo: 2; # iter: 8]:
 pto valutato: 1.6180339887515687
 pto fisso:    1.6180339887498949
```

```
tol = 10^(-9);
maxit = 100;

x = 2;
x0 = 1;
it = 0;
while abs(x-x0)>=tol && it< maxit
    it = it+1;
    x0 = x;
    x = sqrt(x+1);
end

fprintf('METODO DI PUNTO FISSO [# iter: %d]:\n '
  ...'pto valutato: %1.16f \n pto fisso:'
  ...'%1.16f\n',it,x,(1+sqrt(5))/2)
```

```
METODO DI PUNTO FISSO [# iter: 18]:
 pto valutato: 1.6180339889897417
 pto fisso:    1.6180339887498949
```

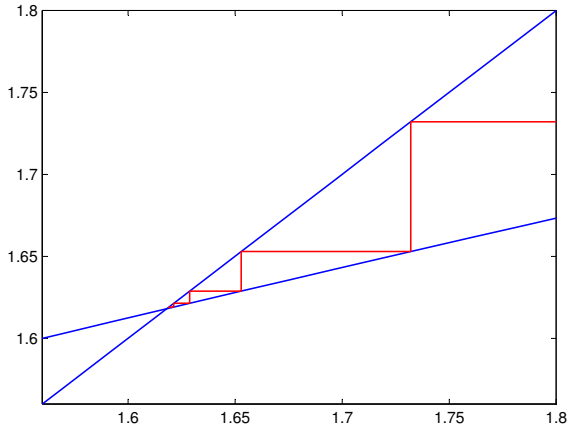

Figura 34: metodo di punto fisso: $x = \sqrt{x+1}$

13 SISTEMI DI EQUAZIONI NON LINEARI

13.1 Metodo di Newton

```
function [x,it] = newton(fname,iacf, xo, tolx, tolf, maxit)
%NEWTON metodo di newton per sistemi di equazioni non lineari
% Dati input:
%  fname    STRING title function della funzione f
%  iacf     STRING title funcion dello iacobiano della funzione f
%  xo       DOUBLE innesco
%  tolx     DOUBLE tolleranza (criterio d'arresto dell'incremento)
%  tolf     DOUBLE tolleranza (criterio d'arresto del residuo)
%  maxit    DOUBLE # massimo iterazioni
%
% Dati output:
%  x        DOUBLE zero della funzione f
%  it       DOUBLE # iterazioni
    x = xo;
    fx = feval(fname,x);
    fprintf('it        ||x||              ||f(x)||\n')
    for i=1:maxit
        [L,U,P] = lu(feval(iacf,x));
        v = U\(L\(P*feval(fname,x)));
        x = x-v;
        fx = feval(fname,x);
        fprintf('%2.f  %.16f  %.16f\n',i,norm(x),norm(fx))
        if(norm(fx)<tolf && norm(v)<tolx)
            break;
        end
    end
    it = i;
    if(it>maxit)
        disp('superato il numero massimo di iterazioni')
    end
end
```

Newton modificato: ciclico (newton_ciclico)

```
function [x,it] = nexton_ciclico(fname,iacf, xo, tolx, tolf, maxit, n)
%NEWTON_CICLICO metodo di newton per sistemi di equazioni non lineari
%               - calcolo dello Iacobiano aggiornato ogni n iterazioni
% Dati input:
%  fname    STRING title function della funzione f
%  iacf     STRING title funcion dello iacobiano della funzione f
%  xo       DOUBLE innesco
%  tolx     DOUBLE tolleranza (criterio d'arresto dell'incremento)
```

```
%  tolf     DOUBLE tolleranza (criterio d'arresto del residuo)
%  maxit    DOUBLE # massimo iterazioni
%  n        DOUBLE ordine del ciclo di aggiornamento dello Iac(f)
%
% Dati output:
%  x        DOUBLE zero della funzione f
%  it       DOUBLE # iterazioni
    x = xo;
    fx = feval(fname,x);
    I = feval(iacf,x);
    for i=1:maxit
        if(mod(i,n)==0)
            I = feval(iacf,x);
            [L,U,P] = lu(feval(iacf,x));
        end
    ...
end
```

Newton modificato: approssimazione alle differenze finite dello iacobiano (`newton_FD`)

```
function [x,it] = nexton_FD(fname, xo, tolx, tolf, maxit,delta)
%NEWTON_FD metodo di newton per sistemi di equazioni non lineari
%          - stima FD dello Iacobiano
% Dati input:
%  fname    STRING title function della funzione f
%  iacf     STRING title funcion dello iacobiano della funzione f
%  xo       DOUBLE innesco
%  tolx     DOUBLE tolleranza (criterio d'arresto dell'incremento)
%  tolf     DOUBLE tolleranza (criterio d'arresto del residuo)
%  maxit    DOUBLE # massimo iterazioni
%  delta    DOUBLE incremento sulle x rapporto incrementale
%
% Dati output:
%  x        DOUBLE zero della funzione f
%  it       DOUBLE # iterazioni
...
for i=1:maxit
    % Approssimazione FD dello Jacobiano.
    n = length(x);
    Jc = zeros(n,n);
    for k=1:n;
        tk = x;
        tk(k) = x(k) + delta(k);
        Jc(:,k) = (feval(fname,tk) - fx)/delta(k);
        ...
```

13.2 *esercizio 1*

Determinare le soluzioni del sistema

$$\begin{cases} 2x - \cos y = 0 \\ 2y + \sin x = 0 \end{cases}$$

```
fprintf('METODO DI NEWTON [x:[0;0]; tol:10^(-10)]:\n')
[x,it] = nexton('funz_1','iacf_1',[0;0],1e-10,1e-10,100);
fprintf('\nit: %d\n',it)
fprintf('x:\n')
disp(x)

fprintf('METODO DI NEWTON [x:[10;-10]; tol:10^(-10)]:\n')
[x,it] = nexton('funz_1','iacf_1',[10;-10],1e-10,1e-10,100);
fprintf('\nit: %d\n',it)
fprintf('x:\n')
disp(x)
```

```
METODO DI NEWTON [x:[0;0]; tol:10^(-10)]:
it        ||x||              ||f(x)||
 1  0.5590169943749475  0.9799187839785651
 2  0.5859762453165357  0.1326585906784402
 3  0.5588265210294080  0.1069136207679839
 4  0.5385422897650281  0.0124140158865710
 5  0.5400853870863919  0.0027294412450861
 6  0.5396033314918051  0.0002749623559290
 7  0.5396505826637718  0.0000305800404303
 8  0.5396452812648787  0.0000032900554331
 9  0.5396458511632167  0.0000003550905736
10  0.5396457896492034  0.0000000383106109
11  0.5396457962858643  0.0000000041334792
12  0.5396457955698086  0.0000000004459751
13  0.5396457956470663  0.0000000000481178
14  0.5396457956387307  0.0000000000051916

it: 14
x:
    4.864051546662073e-001
   -2.337255019562514e-001

METODO DI NEWTON [x:[10;-10]; tol:10^(-10)]:
it        ||x||              ||f(x)||
 1  8.1732654533225073  17.7132089808332720
 2  1.2723682650755930  1.9660946045409144
 3  0.7989651806433054  0.7009896058391094
```

```
 4  0.5731897299566999  0.0832848508728385
 5  0.5446896559856462  0.0302082204400129
 6  0.5392250308059864  0.0031085134886249
 7  0.5397164040321368  0.0004469190586639
 8  0.5396384594703583  0.0000470089939258
 9  0.5396465927043308  0.0000050965252738
10  0.5396457097030372  0.0000005496093694
11  0.5396458049122508  0.0000000593024461
12  0.5396457946390865  0.0000000063983078
13  0.5396457957474852  0.0000000006903358
14  0.5396457956278962  0.0000000000744826
15  0.5396457956407990  0.0000000000080361

it: 15
x:
    4.864051546654787e-001
   -2.337255019625432e-001
```

13.3 *esercizio 2*

Determinare le soluzioni del sistema

$$\begin{cases} x^2 + y^2 - 4 = 0 \\ xy - 1 = 0 \end{cases}$$

```
fprintf('METODO DI NEWTON [x:[1;-5]; tol:10^(-10)]:\n')
[x,it] = nexton('funz_2','iacf_2',[1;-5],1e-10,1e-10,40);
fprintf('\nit: %d\n',it)
fprintf('x:\n')
disp(x)

fprintf('METODO DI NEWTON_CICLICO [x:[1;-5]; tol:10^(-10)]:\n')
[x,it] = nexton_ciclico('funz_2','iacf_2',[1;-5],1e-10,1e-10,40,3);
fprintf('\nit: %d\n',it)
fprintf('x:\n')
disp(x)

fprintf('METODO DI NEWTON_FD [x:[1;-5]; tol:10^(-10)]:\n')
[x,it] = nexton_FD('funz_2', [1;-5], 1e-10, 1e-10, 40,[-0.001;0.001]);
fprintf('\nit: %d\n',it)
fprintf('x:\n')
disp(x)
```

```
METODO DI NEWTON [x:[1;-5]; tol:10^(-10)]:
it        ||x||              ||f(x)||
 1  2.9656599415457068  5.0602218834971806
 2  2.2008379177897557  0.9259283158674987
 3  2.0222758130478322  0.1001154352614774
 4  2.0004591227494606  0.0020534943619746
 5  2.0000002104551950  0.0000009411842924
 6  2.0000000000000444  0.0000000000001977
 7  1.9999999999999998  0.0000000000000001

it: 7
x: -5.176380902050415e-001
   -1.931851652578137e+000

METODO DI NEWTON_CICLICO [x:[1;-5]; tol:10^(-10)]:
it        ||x||              ||f(x)||
 1  2.9656599415457068  5.0602218834971806
 2  2.5263471165845166  2.5567864499881039
 3  2.0893204739858544  0.4060271874580891
 4  2.0329480232131285  0.1484651736999428
 5  2.0134893637294904  0.0605257296831855
 6  2.0001737205277395  0.0007769355562222
 7  2.0000044891702529  0.0000200762022275
 8  2.0000001167603352  0.0000005221681081
 9  2.0000000000000138  0.0000000000000615
10  1.9999999999999996  0.0000000000000004

it: 10
x: -5.176380902050416e-001
   -1.931851652578136e+000

METODO DI NEWTON_FD [x:[1;-5]; tol:10^(-10)]:
it        ||x||              ||f(x)||
 1  2.9653696552875357  5.0586265571284894
 2  2.2005611568863928  0.9248389231516573
 3  2.0221582222197880  0.0996795325015703
 4  2.0004417530039542  0.0019895004357173
 5  1.9999998841501676  0.0000006143673369
 6  1.9999999999696518  0.0000000001213922
 7  2.0000000000000102  0.0000000000000426

it: 7
x: -5.176380902050384e-001
   -1.931851652578148e+000
```

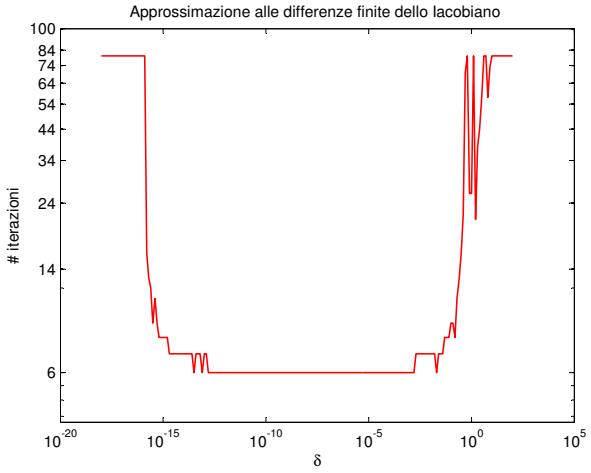

Figura 35: # iterazioni del metodo di Newton con stima dello Iacobiano alle differenze finite, in funzione di δ t.c. $\frac{\partial f_j(x)}{\partial x_i} \sim \frac{f_j(x_i+\delta v_i)-f_j(x_i)}{\delta}$. Se δ è troppo grande l'approssimazione dello Iacobiano può risultare insufficiente; se sono troppo piccoli può occorrere il fenomeno della cancellazione numerica

Parte V
APPROSSIMAZIONE DI DATI E FUNZIONI

14 INTERPOLAZIONE E FUNZIONI SPLINE

14.1 Interpolazione polinomiale

Forma di Lagrange dell'interpolatore polinomiale

```
function [pval] = interpolL(x,y,x1)
%INTERPOLL valuta in x1 l'interpolatore di Lagrange
%          che interpola y nei nodi x
    n = length(x);
    for k=1:n
        p = plagr(x,k);
        L(k,:) = polyval(p,x1);
    end
    pval=y*L;
end

function [p] = plagr(xnodi,k)
%PLAGR Calcola il polinomio di Lagrange di grado k sui nodi xnodi
    if(k==1)
        xzeri = xnodi(2:end);
    else
        xzeri = [xnodi(1:k-1) xnodi(k+1:end)];
    end
    p = poly(xzeri);
    p = p/polyval(p,xnodi(k));
end
```

Forma di Newton dell'interpolatore polinomiale

```
function [pval] = interpolN(x,y,x1)
%INTERPOLN valuta in x1 l'interpolatore di Lagrange
%          in forma di Newton che interpola y nei nodi x
    c = diffdiv(x,y);
    n = length(c);
    pval=c(n)*ones(size(x1));
    for k = n-1:-1:1
        pval=c(k)+(x(k)-x1).*pval;
    end
end
```

```
function [c] = diffdiv(x,y)
%DIFFDIV Differenze divise
% Dati input:
%  x        DOUBLE nodi
%  y        DOUBLE valutazione dei nodi
%
% Dati output:
%  c        DOUBLE coefs della forma di Newton
    n=length(x);
    D(:,1)=y(:);
    for j=2:n
        for i=j:n
            D(i,j) = (D(i-1,j-1)-D(i,j-1))/(x(i)-x(i-j+1));
        end
    end
    c = diag(D);
end
```

14.2 *esercizio 1*

Tracciare i grafici dei polinomi di Lagrange associati ai nodi:

$$\{x_i\}_{i=0,\dots 4} = 0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1$$

$$\{x_i\}_{i=0,\dots 3} = -1, -0.7, 0.5, 2$$

```
a = [0 1/4 1/2 3/4 1];
b = [-1 -0.7 0.5 2];

x1 = [0:0.01:1];
x2 = [-1:0.01:2];

% nodi A
figure
set(0,'DefaultAxesColorOrder',[0 0 0],...
    'DefaultAxesLineStyleOrder','-|-.|--|:')
subplot(1,2,1)
hold all
plot(a,zeros(1,5),'.')
colors = lines(5);
for k=1:5
    plot(x1,polyval(plagr(a,k),x1))
end
title('Polinomi di Lagrange: nodi A')
axis square
```

```
% nodi B
subplot(1,2,2)
hold all
plot(b,zeros(1,4),'.')
for k=1:4
    plot(x2,polyval(plagr(b,k),x2))
end
title('Polinomi di Lagrange: nodi B')
axis square
```

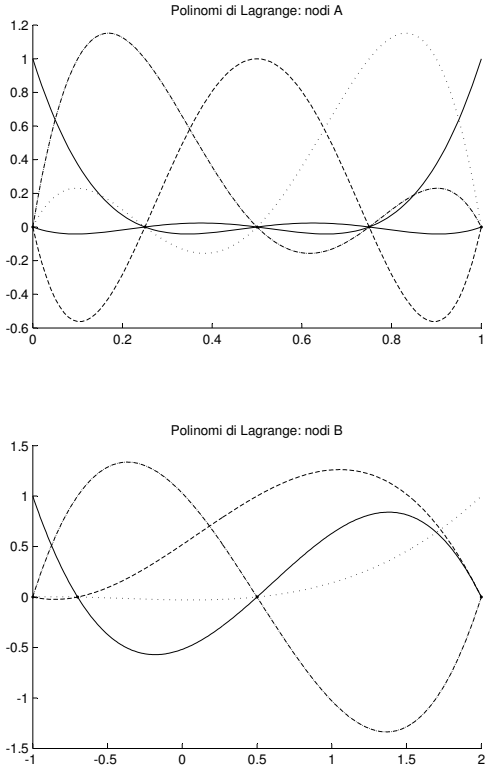

Figura 36

14.3 *esercizio 5*

Fornire un'approssimazione della costante di Lebesgue per nodi equispaziati e nodi di Chebyshev, ove per costante di Lebesgue si intenda

$$\Lambda = \left\| \sum_{j=0}^{n} |L_j^{(n)}| \right\|_\infty$$

e per nodi di Chebyshev

$$x_i = \frac{a+b}{2} - \frac{b-a}{2} \cos\left(\frac{2(i-1)+1}{2(n+1)}\pi\right) \qquad \forall\, i = 1, \dots n+1$$

```
for n = 1:20

    xE = linspace(-1,1,n+1); % nodi equispaziati
    for i=1:n+1
        xC(i) = cos((2*(i-1)+1)*pi/(2*(n+1))); % nodi di Chebyshev
    end

    x = linspace(-1,1,1000);
    y = zeros(size(x));
    for k = 1:n
        y = y+abs(polyval(plagr(xE,k),x));
    end
    lambdaE(j) = max(y); % costante di Lebesgue per nodi equispaziati

    y = zeros(size(x));
    for k = 1:n
        y = y+abs(polyval(plagr(xC,k),x));
    end

    lambdaC(j) = max(y); % costante di Lebesgue per nodi di Chebyshev
end
```

Osservazioni

Sia $f \in \mathcal{C}^0([a,b])$, X una matrice di interpolazione di dimensione n (i.e. n = # nodi), p_n il polinomio interpolante, E l'errore di interpolazione del generico punto x ed E^* l'errore di approssimazione del polinomio di miglior approssimazione p_n^*,

$$E_{n,\infty}(X) = \|f - p_n(x)\|_\infty$$

$$E^*_{n,\infty}(X) = \|f - p_n^*(x)\|_\infty$$

allora

$$E_{n,\infty}(X) \leq (1 + \Lambda_n(X)) E^*_{n,\infty}(X) \quad n \in \mathbb{N}$$

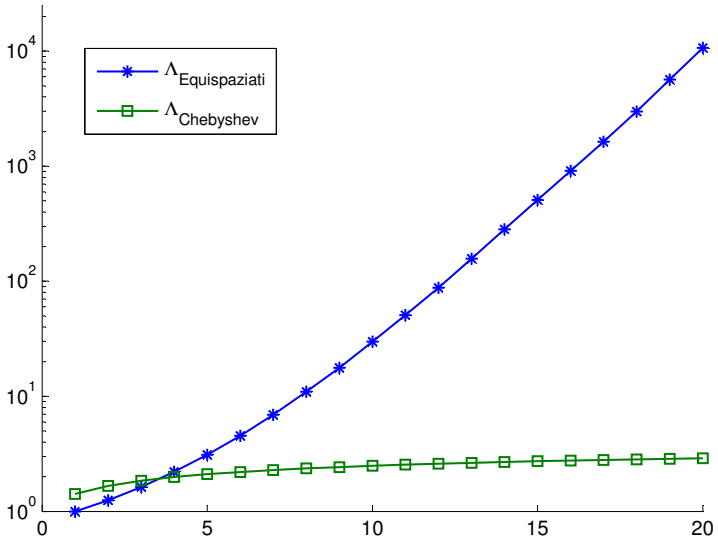

Figura 37: confronto tra la costante di Lebesgue degli zeri del polinomio di Chebyshev e di una partizione equispaziata dell'intervallo [a,b]: *la costante di Lebesgue è misura della stabilità dell'interpolazione polinomiale.*

L'errore d'interpolazione è dominato da una quantità intrinseca del problema e da una dipendente unicamente dalla scelta dei nodi. Per migliorare la stabilità dell'algoritmo di interpolazione polinomiale occorrerà scegliere X tale da minimizzare $\Lambda_n(X)$. Si può mostrare che assegnata una funzione tale minimo esiste, ma solo in rari casi è possibile esplicitarne gli elementi. Si dimostra anche (teorema di Faber) che, assegnata una matrice di interpolazione X, esiste sempre una funzione continua tale che la successione dei polinomi interpolanti di n-esimo grado non converga uniformemente alla funzione. Infatti, la costante di Lebegue è minorata da una funzione che diverge per $n \to \infty$

$$\Lambda_n(X) > \frac{2}{\pi} log(n+1) - C \qquad \forall n \in \mathbb{N}$$

La stima della costante di Lebesgue (prodotta approssimando la norma infinito di f al massimo della valutazione di $|f|$ in un opportuno campionamento dell'intervallo [a,b]) è coerente con i seguenti asserti:

$$\Lambda_{n,Equispaziati}(X) \sim \frac{2^{n+1}}{en \log n} \quad n \to \infty$$

$$\Lambda_{n,Chebyshev}(X) \leq \frac{2}{\pi} log(n+1) + 1 \qquad \forall n \in \mathbb{N}$$

14.4 *esercizio 6*

Interpolare la funzione $\sin(2\pi x)$ con 22 nodi equispaziati x_i nell'intervallo $[-1, 1]$. Perturbare le ordinate dei nodi di un $\epsilon = 0.0002$`*randn(1,22)` e calcolare il polinomio interpolante nei nodi perturbati.
Agli estremi dell'intervallo di interpolazione la differenza fra i polinomi interpolanti è più elevata ($\|\tilde{p}(x) - p(x)\|_\infty \sim 3.8 - 3.9$; $\Lambda_{21} \sim 2.918 \cdot 10^9$)

```
f = inline('sin(2*pi.*x)');
xE = linspace(-1,1,22);
y1 = f(xE);
e = 0.0002*randn(1,22);
y2 = y1+e;

figure
hold all
x = linspace(-1,1,10000);
plot(xE,y1,'o','LineWidth',1)
plot(x,interpolN(xE,y1,x),'r','LineWidth',1)
plot(x,interpolN(xE,y2,x),'--','LineWidth',1)

err = abs(interpolN(xE,y1,x)-interpolN(xE,y2,x));
figure
semilogy(x,err)
```

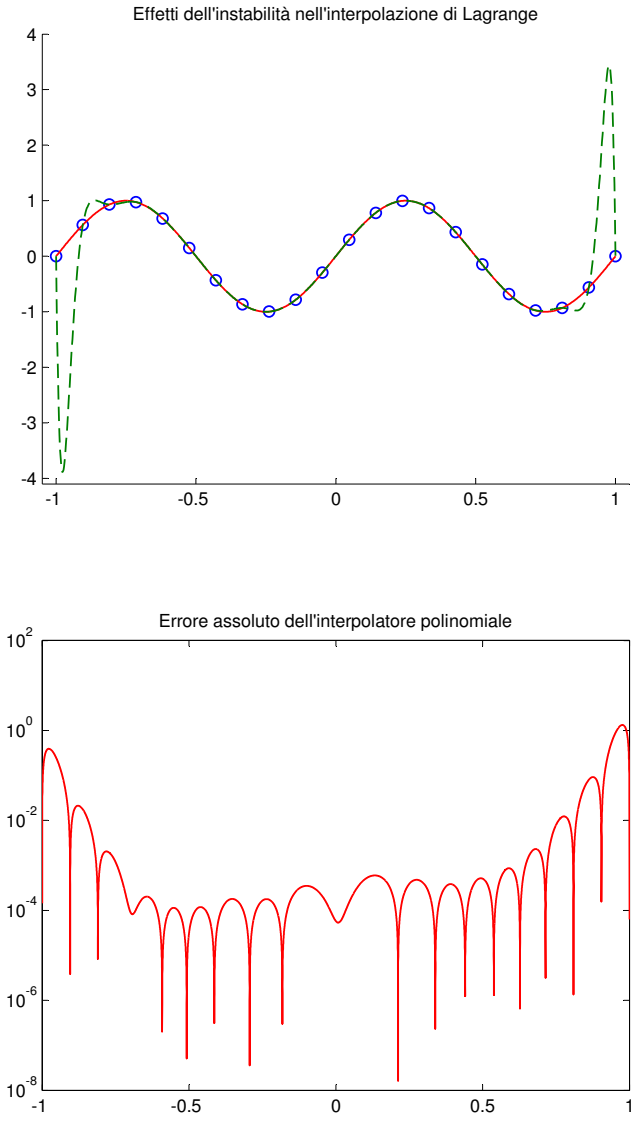

Figura 38: effetti dell'instabilità dell'interpolazione di Lagrange ed errore assoluto ad esso associato

14.5 *esercizio 7*

Interpolare $f(x)$ in 3,5,7,9,11,13 nodi equispaziati e in altrettanti nodi di Chebyshev e calcolare l'errore assoluto di interpolazione.

1. $f_1(x) = \frac{1}{1+x^2}$ $\quad [-10, 10]$ (controesempio di Runge)
2. $f_2(x) = \sinh(x)$ $\quad [-2, 2]$
3. $f_3(x) = \frac{1}{(x-0.3)^2+0.01} + \frac{1}{x^2+0.04} - 6$ $\quad [-1, 3]$

Il codice seguente relativo al controesempio di Runge è adattato alle funzioni successive ed integrato con il calcolo della spline not a knot.

```
    yyS = ppval(spline(xE,f(xE)),xx);
    errS = abs(yyS-yy); % err assoluto not-a-knot

a = -10;
b = 10;
j=1;
n = 9;
figure
for n = [3,5,7,9,11,13]
    xE = linspace(a,b,n); % nodi equispaziati
    xC = zeros(size([1:n])); % nodi Chebyshev
    for i=1:n
        xC(i)=(a+b)/2-(b-a)/2*cos((2*(i-1)+1)*pi/(2*(n)));
    end
    xx = linspace(a,b); % partizione di [a,b]

    f = inline('1./(1+x.^2)');

    yyE = interpolN(xE,f(xE),xx); % interpolazione equispaziata
    yyC = interpolN(xC,f(xC),xx); % interpolazione Chebyshev
    yy = f(xx); % valutazione della funzione

    errE = abs(yyE-yy); % err relativo equispaziati
    errC = abs(yyC-yy); % err relativo Chebyshev


    subplot(6,2,2*j-1)
    plot(xE,f(xE),'.b','LineWidth',2)
    hold on
    plot(xC,f(xC),'.','LineWidth',2,'Color',[0 0.498039215803146 0])
    plot(xx,yyE,'--b','LineWidth',1)
    plot(xx,yyC,'-.','LineWidth',1,'Color',[0 0.498039215803146 0])
    plot(xx,yy,'r','LineWidth',1)
    title(['interpolatore di Lagrange [# nodi: ',int2str(n),']'])
```

```
    subplot(6,2,2*j)
    plot(xx,errE,'r','LineWidth',1)
    hold on
    plot(xx,errC,'--','LineWidth',1)
    title(['errore assoluto [# nodi: ',int2str(n),']'])
    j=j+1;
end
```

Osservazioni

$f_1(x)$ Agli estremi dell'intervallo di interpolazione l'errore di interpolazione del polinomio interpolante sui nodi di Chebyshev è minore dell'errore di interpolazione su nodi equispaziati, mentre in un intorno di zero la disuguaglianza vale al contrario.

$f_2(x)$ Salvo agli estremi dell'intervallo di interpolazione, l'approssimazione del polinomio interpolante su nodi equispaziati è generalmente affetta da un errore inferiore a quella cui è affetto il polinomio interpolante su nodi di Chebyshev.

$f_3(x)$ I metodi sono equivalenti.

Legenda Conveniamo di rappresentare con linea tratteggiata l'interpolatore di Lagrange su nodi equispaziati e con tratto e punto l'interpolatore di Lagrange su nodi di Chebyshev .

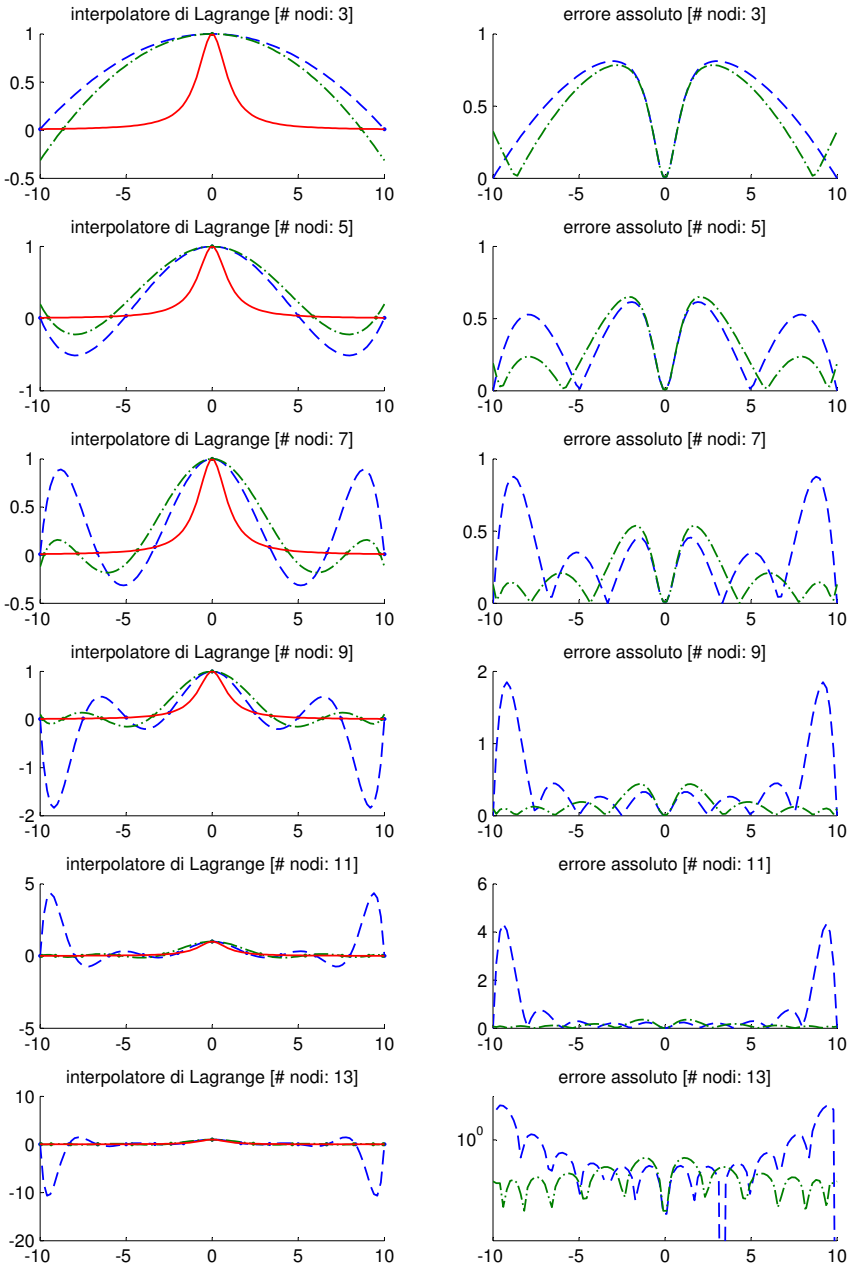

Figura 39: controesempio di Runge: $f(x) = \frac{1}{1+x^2}$

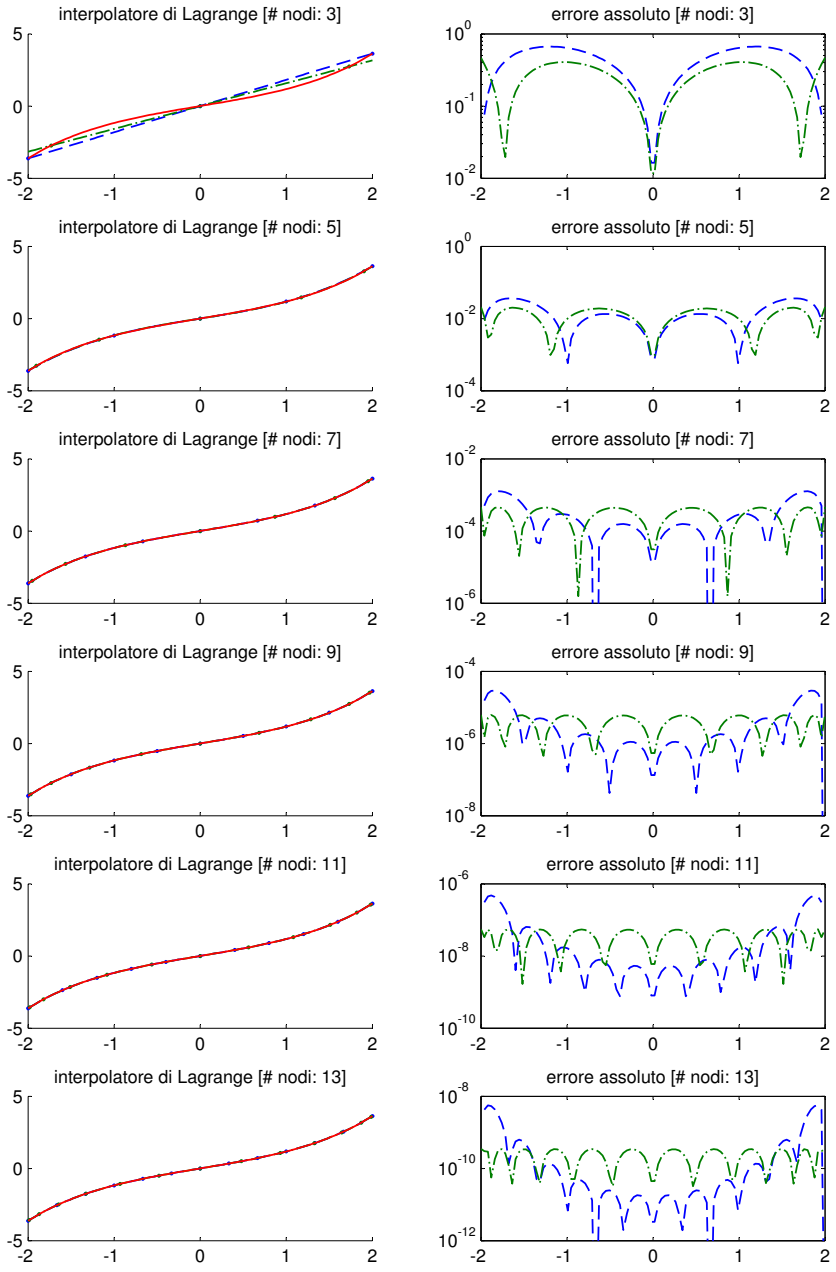

Figura 40: $f(x) = \sinh(x)$

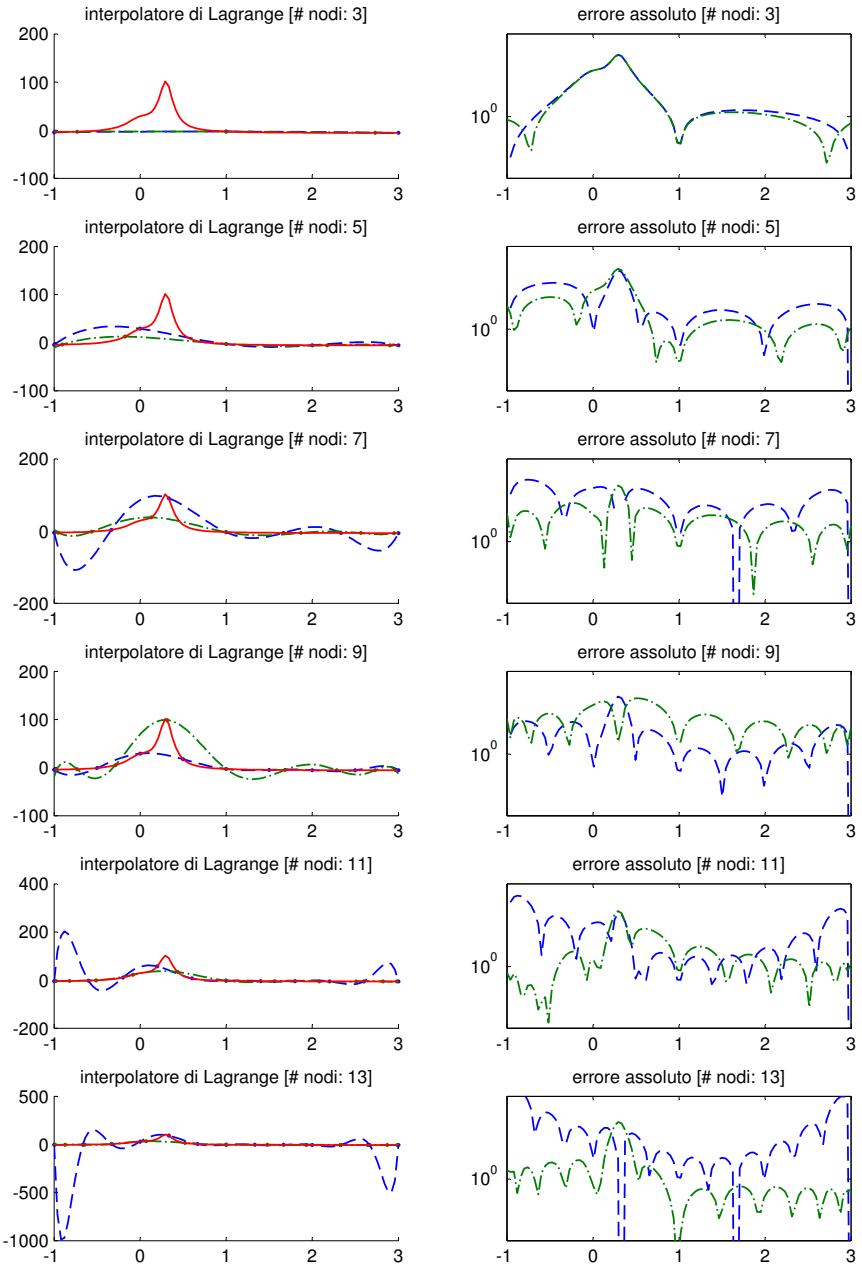

Figura 41: $f(x) = \frac{1}{(x-0.3)^2+0.01} + \frac{1}{x^2+0.04} - 6$

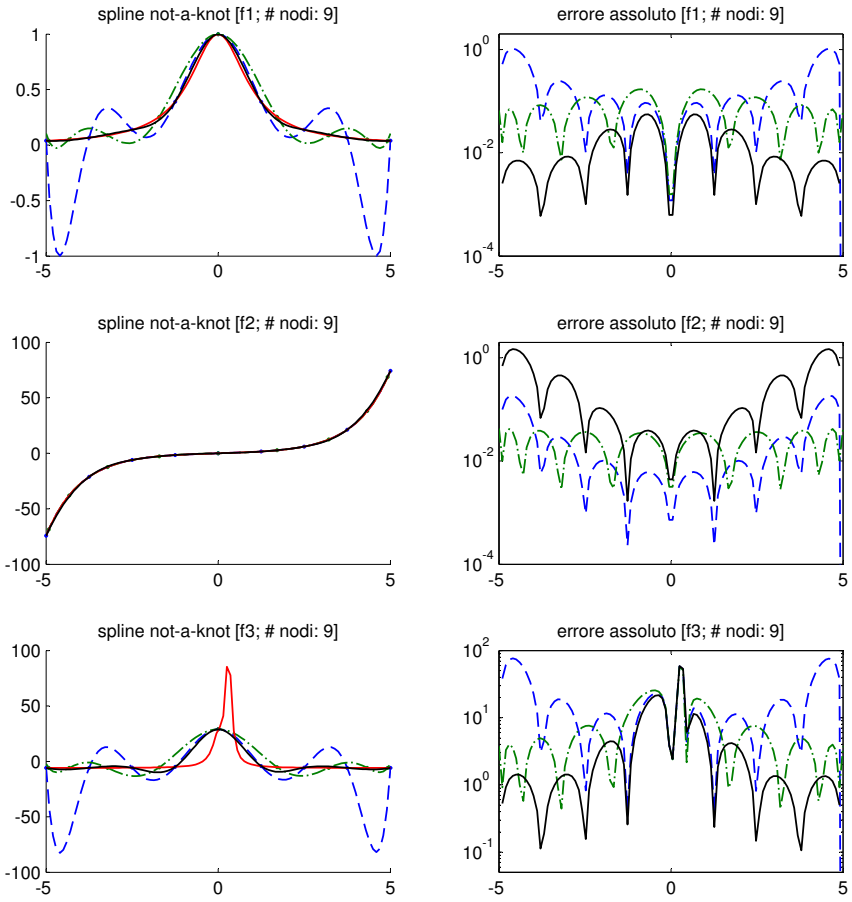

Figura 42: spline not a knot su nodi equispaziati

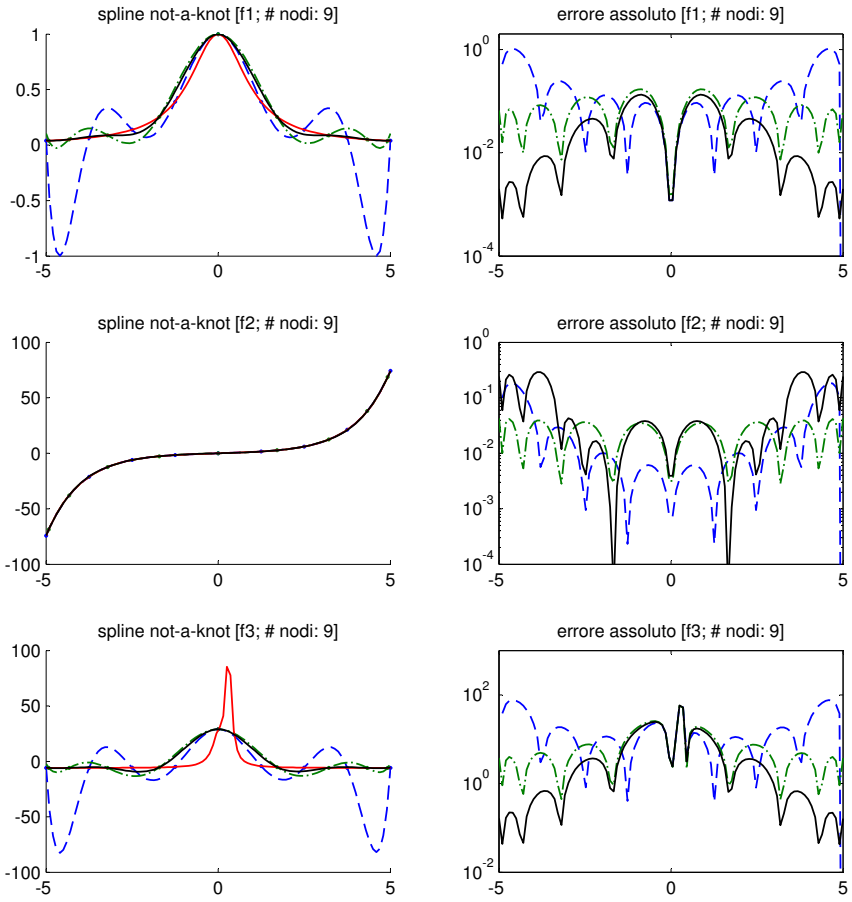

Figura 43: spline not a knot sui nodi di Chebyshev

14.6 *esercizio 8*

La temperatura T in prossimità del suolo varia al variare della concentrazione di acido carbonico e della latitudine L. La temperatura al suolo subisce una variazione dipendente dalla temperatura secondo la seguente tabella:

L	-55	-45	-35	-25	-15	-5	5	15	25	35	45
T	3.7	3.7	3.52	3.27	3.2	3.15	3.15	3.25	3.47	3.52	3.65

Si vuole costruire un modello che descriva la legge T=T(L) anche per latitudini non misurate. Valutare la variazione di temperatura alle latitudini $L = \pm 42$.

```
L = [ -55 -45 -35 -25 -15 -5 5 15 25 35 45 55 65];
T = [3.7 3.7 3.52 3.27 3.2 3.15 3.15 3.25 3.47 3.52 3.65 3.67 3.52];

x = [42;-42];
yE = interpolN(L,T,x);
yS = ppval(spline(L,T),x);
data = [x,yE,yS];
fprintf('INTERPOLAZIONE DATI:\n')
fprintf(' L       T_interpol Lagrange     T_spline\n')
disp(data)

xx = linspace(min(L),max(L));
yyE = interpolN(L,T,xx); % interpolazione polinomiale equispaziata
yyS = ppval(spline(L,T),xx); % spline interpolatoria not-a-knot

figure
plot(L,T,'.b','LineWidth',2)
hold on
plot(xx,yyE,'-.b','LineWidth',1)
plot(xx,yyS,'r','LineWidth',1,'Color',[0 0.498039215803146 0])
title(['var della temp del suolo in funzione della latitudine'])
xlabel('L')
ylabel('T')
```

```
INTERPOLAZIONE DATI:
  L     T_interpol Lagrange        T_spline
  42     3.552289457164020    3.610216958847710
 -42     3.783695425446566    3.663109164430728
```

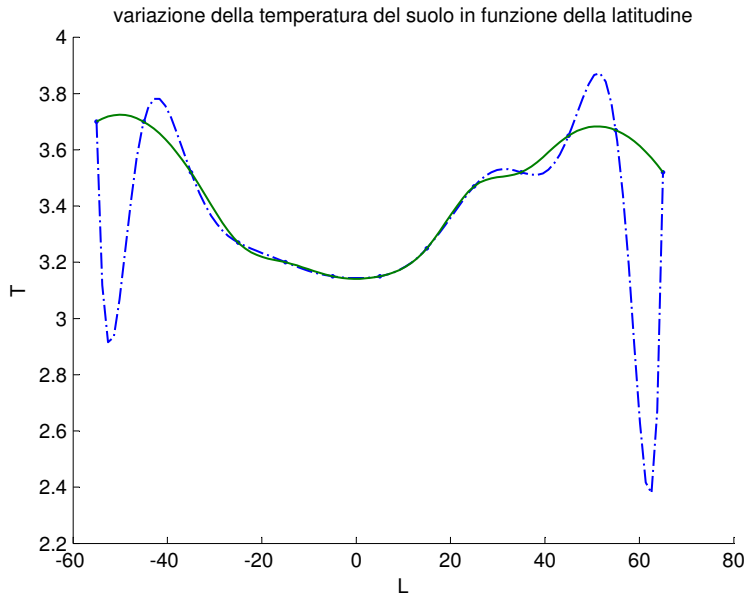

Figura 44: variazione della temperatura del suolo in funzione della latitudine ($[H_2CO_3]$=1.5): confronto tra l'interpolazione polinomiale (tratto punto) e la spline not-a-knot (tratto continuo).

14.7 *esercizio 9*

Considerare dati (x, y) provenienti dalle seguenti funzioni test:

1. $f_1(x) = x^5 - 4x^2 + 2 \qquad [-2, 2]$
2. $f_2(x) = \cos(x) \qquad [0, 2\pi]$
3. $f_3(x) = e^x \cos(4x) \qquad [0, 3]$

Utilizzando rispettivamente 7, 7, 10 nodi. Osservare come la scelta delle condizioni iniziali e finali influenzi l'andamento della spline interpolante dei dati assegnati (spline naturale, periodica, completa, not a knot). Nel caso della spline completa considerare perturbazioni delle derivate agli estremi:

- il valore approssimato fornito dal rapporto incrementale
- valori errati (eventualmente random) scelti arbitrariamente.

```
j=1;
figure
for k = [1,2,3]
    if(k==1)
        f = inline('x^5-4*x^2+2');
        a = -2;
        b = 2;
        n = 7;
        dfa = 64;
        dfb = 96;
    elseif(k==2)
        f = inline('cos(x)');
        a = 0;
        b = 2*pi;
        n = 7;
        dfa = 0;
        dfb = 0;
    else
        f = inline('exp(x)*cos(4*x)');
        a = 0;
        b = 3;
        n = 10;
        dfa = 1;
        dfb = exp(3)*(cos(12)-4*sin(12));
    end
    f = vectorize(f);
    x = linspace(a,b,n); % nodi equispaziati
    xx = linspace(a,b); % partizione di [a,b]
    yy = f(xx); % valutazione della funzione
```

```
    yL = interpolN(x,f(x),xx); % interpolazione equispaziata
    yV = ppval(csape(x,f(x),'v'),xx); % spline naturale
    yP = ppval(csape(x,f(x),'periodic'),xx); % spline periodica
    yC = ppval(csape(x,[dfa f(x) dfb],'clamped'),xx); % spline compl
    yNaK = ppval(csape(x,f(x),'not-a-knot'),xx); % spline not-a-knot

    errL = abs(yL-yy)'; % err ass interpolatore di Lagrange
    errV = abs(yV-yy)'; % err ass spline naturale
    errP = abs(yP-yy)'; % err ass spline periodica
    errC = abs(yC-yy)'; % err ass spline completa
    errNaK = abs(yNaK-yy)'; % err ass spline not-a-knot

    subplot(3,1,j)
    semilogy(xx,errL,':b','LineWidth',1)
    hold on
    semilogy(xx,errV,'-.b','LineWidth',1)
    semilogy(xx,errP,'--r','LineWidth',1)
    semilogy(xx,errC,':k','Color',[0 0.498039215803146 0])
    semilogy(xx,errNaK,':k','LineWidth',1)
    legend('interpolatore di Lagrange', 'spline naturale',...
        'spline periodica','spline completa','spline not-a-knot')
    hold off

    j=j+1;
end
```

(variante spline completa)

```
    yyC = ppval(csape(x,[dfa f(x) dfb],'clamped'),xx); % spline compl
    errC = abs(yyC-yy); % err ass spline completa
    semilogy(xx,errC,'r','LineWidth',1);

    dfa = (f(x(1))-f(x(2)))/(x(1)-x(2));
    dfb = (f(x(end))-f(x(end-1)))/(x(end)-x(end-1));
    yyCRapp = ppval(csape(x,[dfa f(x) dfb],'clamped'),xx);
         % spline completa (rapporto incrmentale)
    errCRapp = abs(yyCRapp-yy); % err ass spline completa rapp incr
    semilogy(xx,errRapp,'--k','LineWidth',1)

    for k = 1:4
        dfa = 10*randn(1);
        dfb = 10*randn(1);
        yyCRand = ppval(csape(x,[dfa f(x) dfb],'clamped'),xx);
            % spline completa random
        errCRand = abs(yyCRand-yy); % err ass spline completa random
        semilogy(xx,errCRand,':b','LineWidth',1)
    end
```

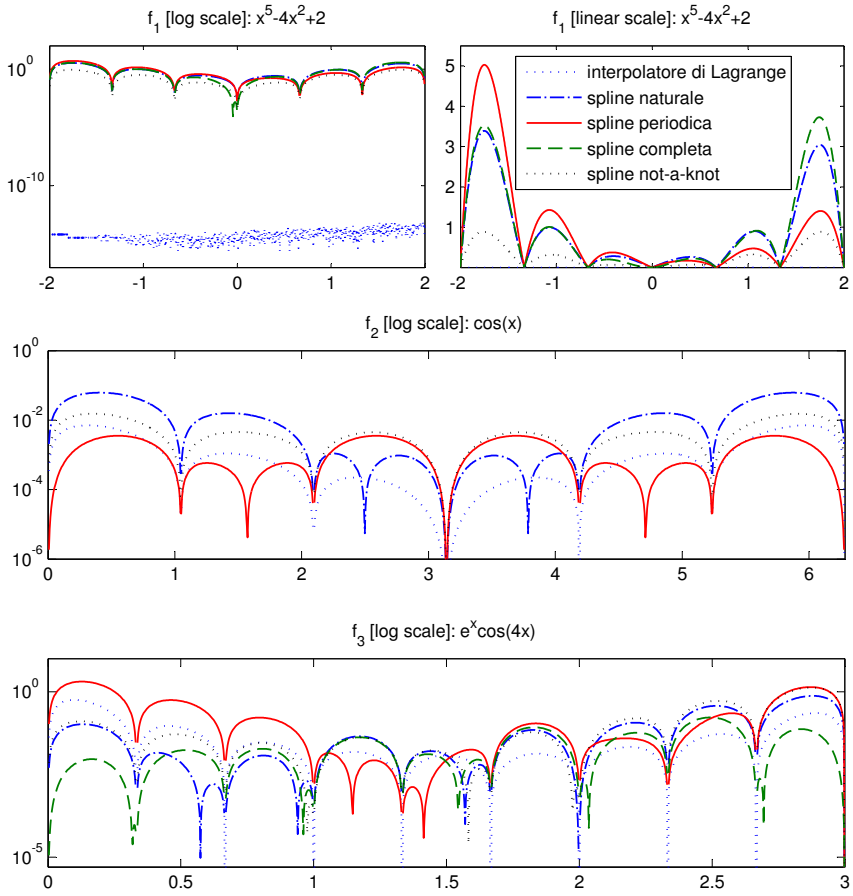

Figura 45: confronto dell'errore assoluto d'interpolazione per le seguenti strategie interpolatoria: interpolazione di Lagrange, spline naturale, periodica, completa, not a knot.

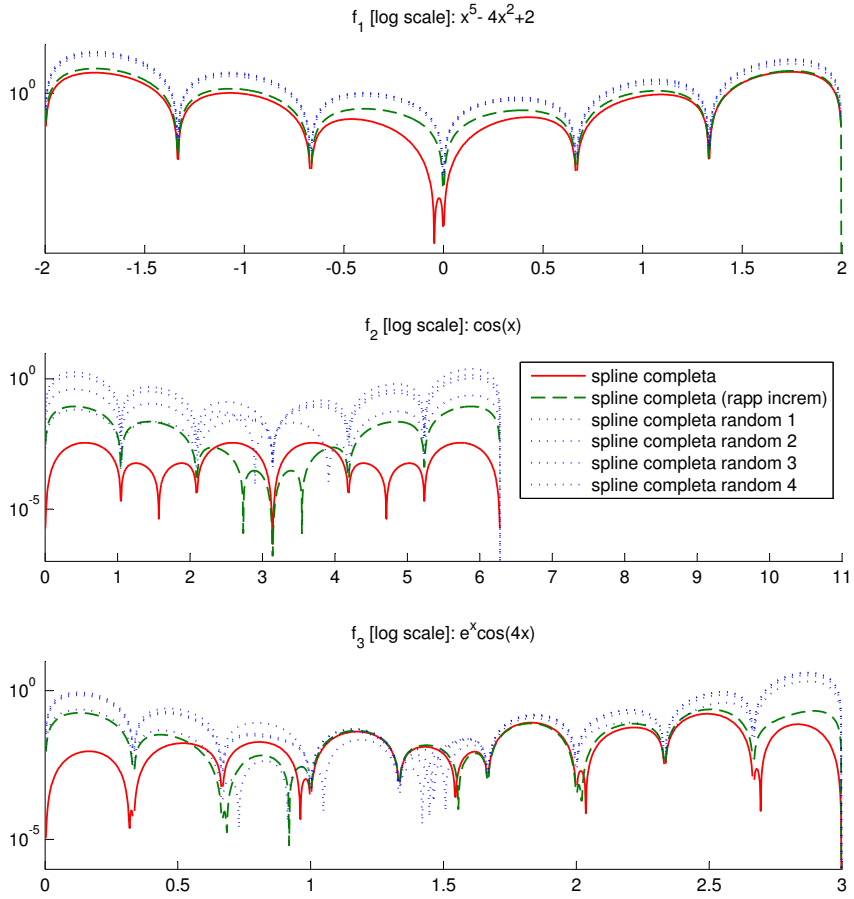

Figura 46: perturbazioni dell'interpolazione attraverso spline complete

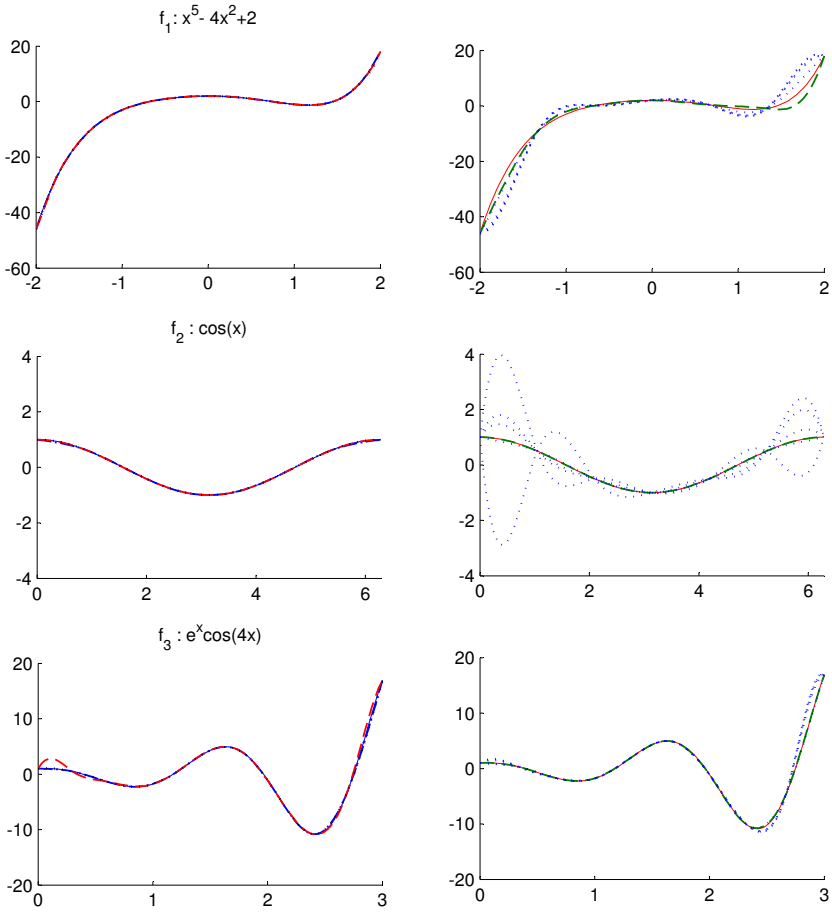

Figura 47: curve interpolanti

Osservazioni

$f_1(x)$ La funzione è un polinomio. L'operatore che associa ad ogni coppia (f, X_n), ove $f \in \mathcal{C}[a,b]$ e X_n matrice di interpolazione $n \times n$ su $[a,b]$, è una proiezione dello spazio delle funzioni continue sul sottospazio dei polinomi di grado n, i.e. l'interpolatore di grado 7 di un polinomio di grado 5 è il polinomio stesso. L'errore assoluto (nell'aritmetica di macchina) commesso stimando la funzione f_1 con il suo interpolatore è dell'ordine della precisione di macchina.

$f_2(x)$ Soprattutto al bordo, l'errore d'interpolazione della spline periodica è inferiore a quello delle altra strategie d'interpolazione. Spline completa e spline periodica coincidono (le condizioni al bordo ovvero di chiusura coincidono).

$f_3(x)$ I metodi sono equivalenti. L'imposizione delle condizioni di periodicità al bordo per funzioni non periodiche è generalmente peggiorativa.

Agli estremi dell'intervallo di interpolazione l'errore assoluto della spline completa, calcolata nelle derivate degli estremi, è minimo rispetto ogni sua perturbazione. L'approssimazione della funzione attraverso spline interpolante not a knot, adottata dalla function `spline` di MATLAB, è tendenzialmente la migliore (qualora non siano note ulteriori proprietà di regolarità della funzione, i.e. periodicità). In generale, si osserva l'aderenza delle spline alle funzioni interpolanti. La precedente intuizione si rigorizza con il seguente teorema:

> Tra tutte le funzioni $f(x) \in \mathcal{C}^2[a,b]$ che soddisfano le condizioni di interpolazione, la spline cubica naturale minimizza il seguente funzionale
> $$E(x) = \int_a^b [f''(x)]^2 \, dx$$

Il risultato è noto come principio di norma minima, vale anche nel caso in cui al posto delle condizioni naturali, vengano imposte condizioni di derivate prima assegnata agli estremi. Il funzionale $E(x)$ è misura della curvatura di f.

* * *

> *[The] spline functions are the most successful approximating functions for pratical applications so far discovered. The reader may be unaware of the fact that ordinary polynomials are inadeguate in many situations. This is particulary the case when one approximates functions which arise from the physical world rather then from the mathematical world. Functions which express physical relationship are frequently of a disjointed or disassociated nature. This is to say that their behaviour in one region may be totally unrelated to their behaviour in another region. Polynomials along with most other mathematical functions, have just the opposite property. Namely, their*

behaviour in a small region determines their behaviour everywhere. Splines do not suffer this handicap since they are defined piecewise, yet, for $n \geq 3$, they represent nice, smooth curves in the physical world.

[J. R. Rice, *The approximation of functions* (1969)]

15 APPROSSIMAZIONE AI MINIMI QUADRATI DISCRETI

15.1 *esercizio 1*

(cfr. esercizio 8) La temperatura al suolo subisce una variazione dipendente dalla temperatura secondo la seguente tabella:

L	-55	-45	-35	-25	-15	-5	5	15	25	35	45
T	3.7	3.7	3.52	3.27	3.2	3.15	3.15	3.25	3.47	3.52	3.65

Si vuole costruire un modello che descriva la legge T=T(L) anche per latitudini non misurate. Confrontare i metodi di interpolazione polinomiale, spline ed approssimazione ai minimi quadrati discreti. Studiare il polinomio di miglior approssimazione nel senso dei minimi quadrati al variare del grado. Valutare la variazione di temperatura alle latitudini $L = \pm 42$.

```
L = [ -55 -45 -35 -25 -15 -5 5 15 25 35 45 55 65];
T = [3.7 3.7 3.52 3.27 3.2 3.15 3.15 3.25 3.47 3.52 3.65 3.67 3.52];

xx = linspace(min(L),max(L));
yyE = interpolN(L,T,xx); % interpolazione polinomiale equispaziata
yyS = ppval(spline(L,T),xx); % spline interpolatoria not-a-knot
yyM = polyval(polyfit(L,T,5),xx);  % least-squares

figure
hold on
plot(L,T,'.b', 'MarkerSize',6)
plot(xx,yyE,':b','LineWidth',1)
plot(xx,yyS,'--','LineWidth',1,'Color',[0 0.498039215803146 0])
plot(xx,yyM,'r','LineWidth',1)
title(['variazione della temperatura...'])
xlabel('L')
ylabel('T')

%% Grado del polinomio di approx nel senso dei minimi quadrati
figure; j=1;
for k=[1,2,3,4,5,7,10,20]
    yyM = polyval(polyfit(L,T,k),xx);

    subplot(4,2,j)
    plot(L,T,'.b', 'MarkerSize',6)
    hold on
    plot(xx,yyM,'r','LineWidth',1)
    title(['degr.p(x) [least-squares]: ',int2str(k)])
    j=j+1;
```

```
end

%% Cfr interpolazione del dato [42;-42]
x = [42;-42];
fprintf('INTERPOLAZIONE E APPROSSIMAZIONE DATI:\n')
fprintf('L  T_interpol Lagrange  T_spline  T_least-squares\n')
for k=[1,2,3,4,5,7,10,20]
    yE = interpolN(L,T,x);
    yS = ppval(spline(L,T),x);
    yM = polyval(polyfit(L,T,k),x);
    data = [x,yE,yS,yM];
    disp(data)
end
```

```
INTERPOLAZIONE E APPROSSIMAZIONE DATI:
 G    T_interpol Lagrange        T_spline         T_least-squares
 1    3.552289457164020    3.610216958847710    3.457670329670329
      3.783695425446566    3.663109164430728    3.426285714285714

 2    3.552289457164020    3.610216958847710    3.453597902097902
      3.783695425446566    3.663109164430728    3.532562937062938

 3    3.552289457164020    3.610216958847710    3.539559833916084
      3.783695425446566    3.663109164430728    3.504467701048951

 4    3.552289457164020    3.610216958847710    3.642291590596903
      3.783695425446566    3.663109164430728    3.604386695492009

 5    3.552289457164020    3.610216958847710    3.632878430253497
      3.783695425446566    3.663109164430728    3.638510960917833

Warning: Polynomial is badly conditioned. [...]

 7    3.552289457164020    3.610216958847710    3.628174210354934
      3.783695425446566    3.663109164430728    3.651822595809541

Warning: Polynomial is badly conditioned. [...]

10    3.552289457164020    3.610216958847710    3.601313402631252
      3.783695425446566    3.663109164430728    3.708842519647890

Warning: Polynomial is not unique; degree >= number of data points.

20    3.552289457164020   3.610216958847710     0.007875167773367
      3.783695425446566    3.663109164430728    9.938644366900792
```

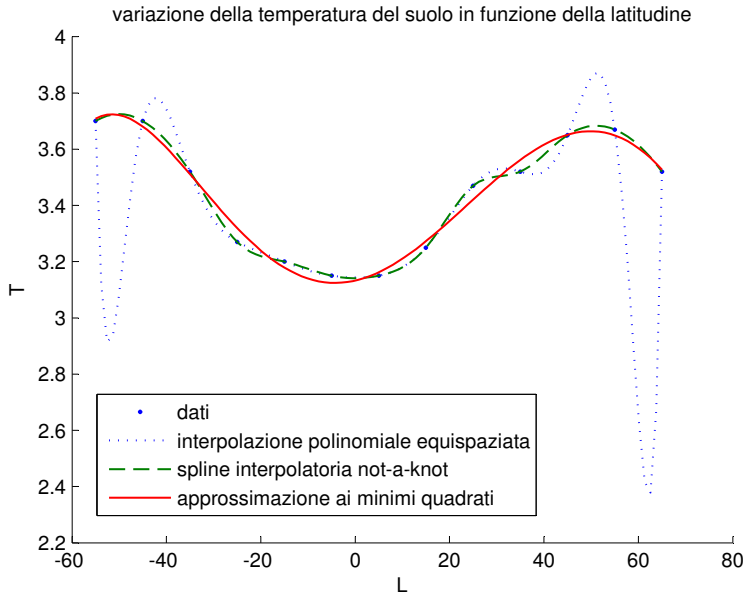

Figura 48: variazione della temperatura del suolo in funzione della latitudine ($[H_2CO_3]$=1.5): confronto tra l'interpolazione polinomiale (puntinato), la spline not-a-knot (tratteggiato) e l'approssimazione polinomiale nel senso dei minimi quadrati di grado 5 (tratto continuo).

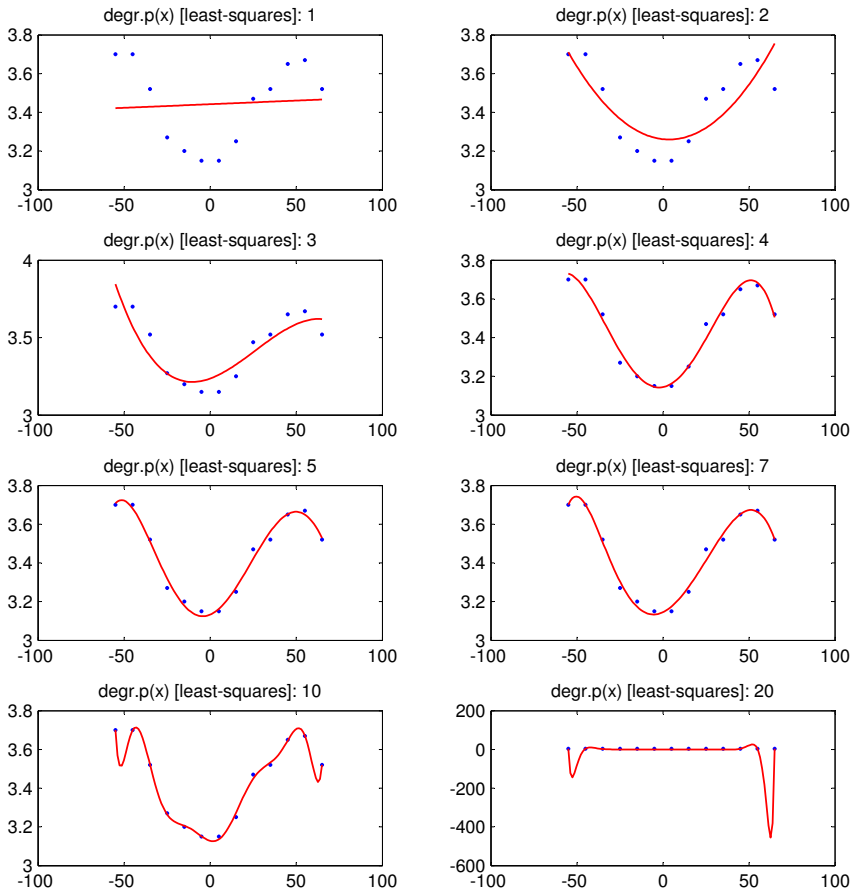

Figura 49: variazione del polinomio di miglior approssimazione nel senso dei minimi quadrati in funzione del grado.

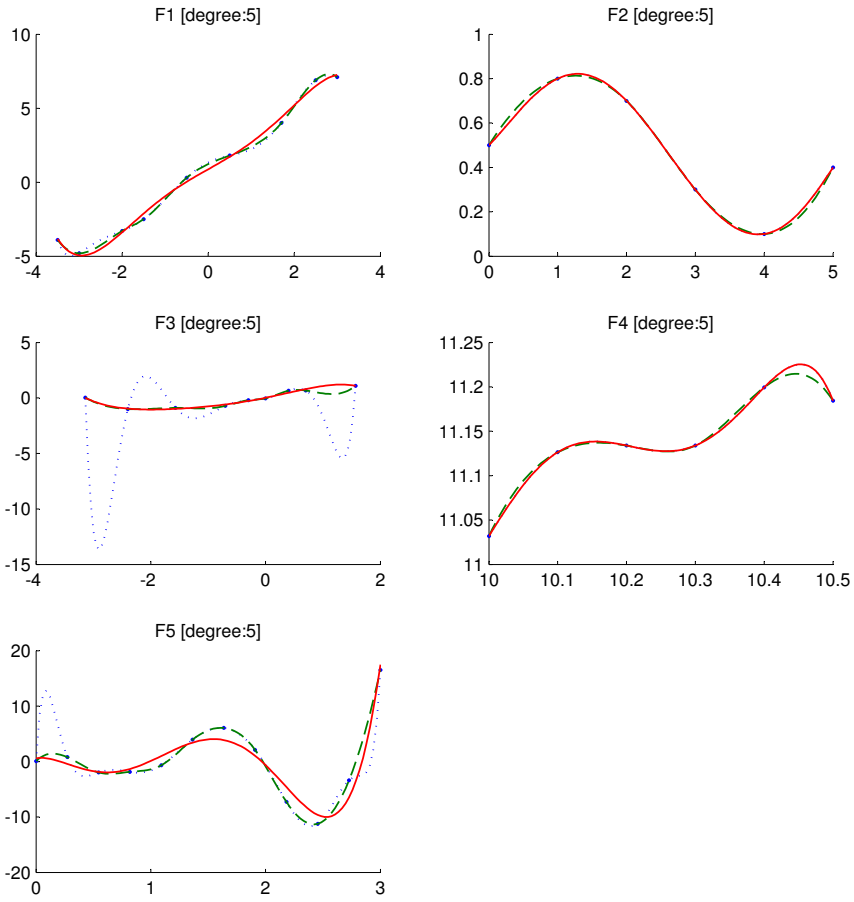

Figura 50: *Trovare un'approssimazione ai minimi quadranti di opportune configurazioni di dati.* Confronto tra l'interpolazione polinomiale (puntinato), la spline not-a-knot (tratteggiato) e l'approssimazione polinomiale nel senso dei minimi quadrati di grado 5 (tratto continuo)

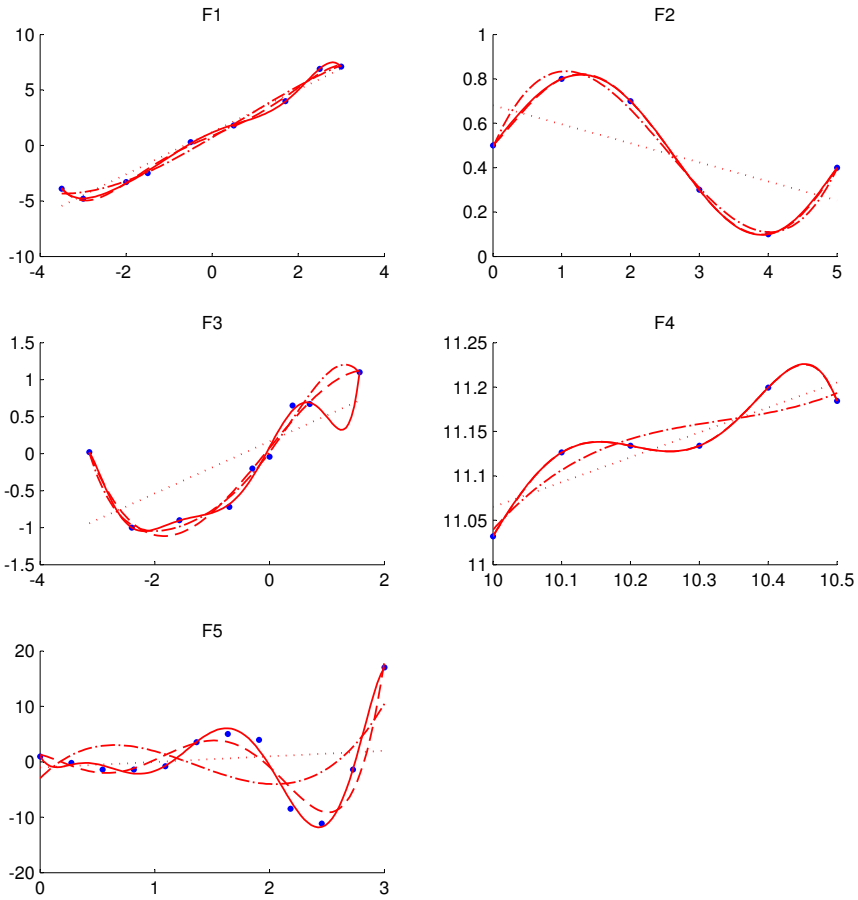

Figura 51: variazione del polinomio di miglior approssimazione nel senso dei minimi quadrati in funzione del grado.

Parte VI
INTEGRAZIONE NUMERICA

16 INTEGRAZIONE AUTOMATICA

16.1 Integrazione automatica adattiva

```
function [QN,cnt] = quad1(nomefun,a,b,tol)
% QUAD1   Integrazione automatica adattiva (metodo dei trapezi)
%
% Dati input:
%  nomefun  STRING title function della funzione f
%  a        DOUBLE estremo sinistro di integrazione
%  b        DOUBLE estremo destro di integrazione
%  tol      DOUBLE tolleranza
%
% Dati output:
%  QN       DOUBLE numerical integral evaluation
%  cnt      DOUBLE # passi ricorsivi

aliv = a; faliv=feval(nomefun,a);
bliv = b; fbliv=feval(nomefun,b);
QV = (faliv+fbliv)*(b-a)/2; % base ricorsione
[QN,cnt] = quadtr(nomefun,a,b,aliv,bliv,tol,1,QV,faliv,fbliv);
cnt=cnt+2;
end

function [QN,cnt] = quadtr(nomefun,a,b,aliv,bliv,
      ... tol,livello,QV,faliv,fbliv)
livello=livello+1;
cnt=0;
if livello<15 % criterio d'arresto: # ricorsioni
    h=bliv-aliv; cnt=1;
    mezzo=(aliv+bliv)/2;
    fmezzo=feval(nomefun,mezzo);

% - print nodi
    hold on
    plot([aliv bliv],[faliv fbliv],'ro')

% - stima migliore QN1+QN2 (bisezione)
    QN1=(faliv+fmezzo)*h/4;
    QN2=(fbliv+fmezzo)*h/4;
```

```
    if (abs(QN1+QN2-QV)/3 >=(h/(b-a)*tol)) % crit d'arresto: toll
        [QN1,cnt1] = quadtr(nomefun,a,b,aliv,mezzo,
                     ... tol,livello,QN1,faliv,fmezzo);
        [QN2,cnt2] = quadtr(nomefun,a,b,mezzo,bliv,
                     ... tol,livello,QN2,fmezzo,fbliv);
        cnt=cnt1+cnt2+1;
    end
    QN=QN1+QN2;
else
    QN=QV; % integrale del sottointervallo
end
end
```

16.2 `function quad1 e myquad`

Confrontare le strategie di integrazione adattiva trapezi e Simpson per le seguenti funzioni test:

1. $f_1(x) = x^5 + 1 \qquad [0, 1]$
2. $f_2(x) = \sqrt{x} \qquad [0, 1]$
3. $f_3(x) = \sqrt{x} \qquad [0.1, 1.1]$
4. $f_4(x) = \sin x + e^{-10x} \qquad [0, \pi]$
5. $f_5(x) = e^{\sqrt{x}} \sin x + 2x - 4 \qquad [-1, 3]$
6. $f_6(x) = \arctan 10x \qquad [-3, 4]$

Confrontare anche le seguenti e significative funzioni test:

1. $f_7(x) = \sin x \qquad [0, \pi]$
2. $f_8(x) = |\sin x| \qquad [0, 2\pi]$

Nota La `myquad` usa un criterio di arresto del tipo

$$|\texttt{QN-QV}| < tol_{rel}|\texttt{QN}| + tol_{abs}$$

ove `tol`$= [tol_{rel}; tol_{abs}]$ e se in input `tol` è uno scalare viene automaticamente creato un vettore `tol=[tol,0]`. Occorre quindi prestare attenzione al tipo di criterio di arresto che si vuole considerare e al confronto dei costi computazionali. Infatti, la `quad1` usa un criterio di arresto del tipo:

$$\frac{|\texttt{QN-QV}|}{3} < tol\,\frac{\texttt{bliv-aliv}}{\texttt{b-a}}$$

```
INTEGRALE DEFINITO:

       Trapezi             Simpson              toll
  1.0e+002 *

   0.011667323371366   0.011666672825813   0.000001000000000
   0.006666143608010   0.006666655870027   0.000001000000000
   0.007479867509651   0.007480429094061   0.000001000000000
   0.020999714640142   0.021000007602275   0.000001000000000
   2.948720223197691   2.948727765510008   0.000010000000000
   0.015420328573863   0.015420326011703   0.000001000000000

# passi ricorsivi:
         T           S
         71          29
         77         109
         39          25
        305         129
       4563         241
        589         241
```

Osservazioni

$f_{2,3}(x)$ Il numero di passi di ricorsione dipende dalla regolarità della funzione. Eliminando dall'intervallo d'integrazione un intorno di $x = 0$, la funzione risulta lipschitziana: il numero di passi di ricorsione diminuisce (il costo computazionale dei due metodi si inverte[1])

$f_5(x)$ Si nota in generale come a parità di tolleranza il numero di passi ricorsivi dell'integrazione numerica adattiva trapezi è maggiore di quello di Simpson.

$$E_T(f) = -\frac{h^3}{12} f^{(2)}(\xi) \quad h = b - a$$

$$E_S(f) = -\frac{h^5}{90} f^{(4)}(\xi) \quad h = \frac{b-a}{2}$$

$f_7(x)$ Il numero di passi di ricorsione dipende dalla tolleranza assegnata.

$f_7(x)$ Per opportune simmetrie della funzione integranda il metodo dei trapezi può fallire: è sufficiente che la funzione ammetta come zeri gli estremi di integrazione e il loro punto medio.

[1] Si intenda come misura del costo computazionale il numero di passi di ricorsione effettuati. Si trascurano in prima approssimazione il numero di valutazioni funzionali effettuate: Simpson effettua circa due valutazione funzionale ad ogni passo contro una valutazione del metodo dei trapezi. Di qui il vantaggio di impiegare successioni annidate di formule di quadratura per un'efficiente implementazione delle strategie di integrazione numerica su calcolatore

```
INTEGRALE DEFINITO (f(x) = sin(x), [0,pi]):

       Trapezi             Simpson              toll
  1.974231601945551   2.000269169948388   0.100000000000000
  1.993570343772339   2.000269169948388   0.010000000000000
  1.999506661370469   2.000016591047936   0.001000000000000
  1.999952020215288   2.000001033369413   0.000100000000000
  1.999993070211637   2.000000064530002   0.000010000000000
  1.999999478549419   2.000000004032257   0.000001000000000
  1.999999952536231   2.000000000015750   0.000000100000000
  1.999999975279255   2.000000000000985   0.000000010000000

# passi ricorsivi:

        T           S
          9           9
         17           9
         57          17
        193          33
        469          65
       1741         129
       6105         513
       8001        1025

INTEGRALE DEFINITO (f(x) = |sin(x)|, [0,2*pi]):

       Trapezi             Simpson              toll
  0.000000000000001   4.009119509968842   0.100000000000000
  0.000000000000001   4.000538339896776   0.010000000000000
  0.000000000000001   4.000002066738826   0.001000000000000
  0.000000000000001   4.000000129060004   0.000100000000000

# passi ricorsivi:

    T     S
    3     9
    3    17
    3    65
    3   129
```

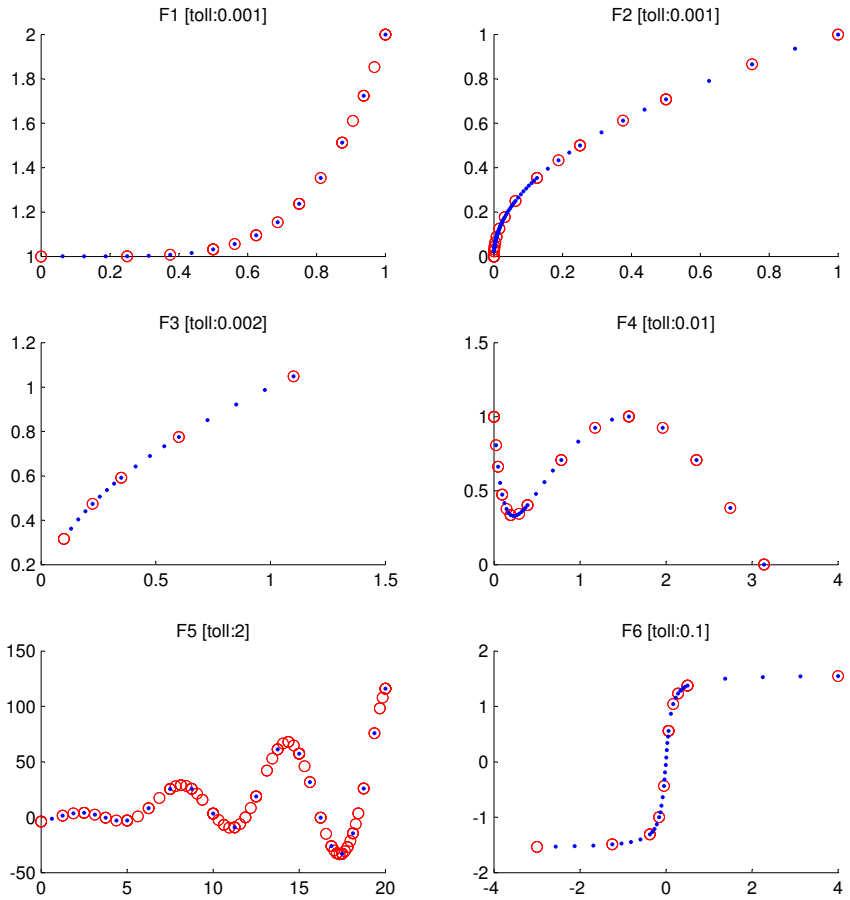

Figura 52: `quad1` (integrazione adattiva trapezi: cerchio) e `myquad` (integrazione adattiva Simpson: punto)

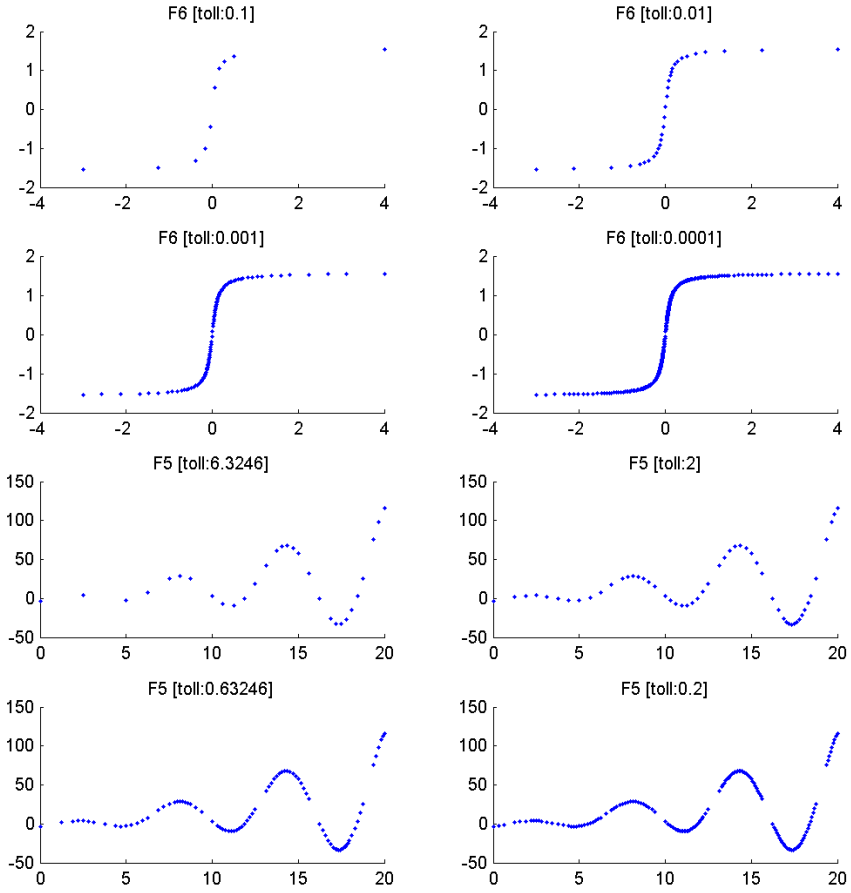

Figura 53: distribuzione dei nodi `quad1` in funzione della tolleranza assegnata

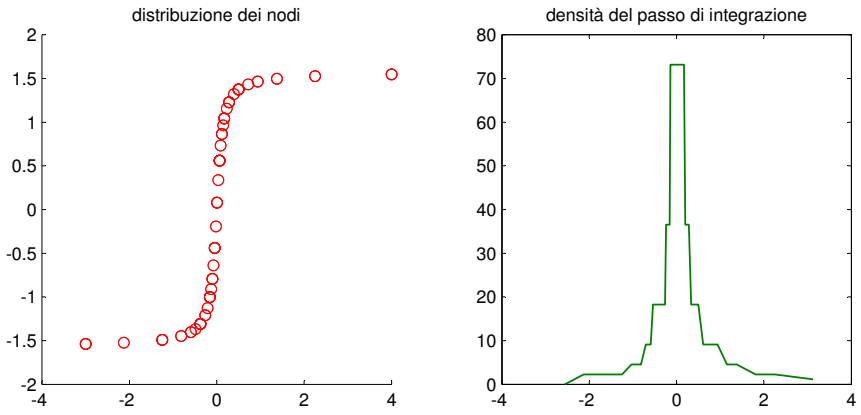

Figura 54: distribuzione dei nodi di quadratura e densità del passo di integrazione nel calcolo di $I(f) = \int_{-3}^{4} \arctan(10x)dx$. *Si nota in generale un aumento della distribuzione dei nodi in corrispondenza dei massimi della derivata.*

Parte VII
EQUAZIONI DIFFERENZIALI ORDINARIE

17 ESERCITAZIONE 13

17.1 Equazioni di Lotka-Volterra e Caduta di un grave

Risolvere con il solutore `ode45()` il sistema dinamico di crescita delle popolazioni di Lotka-Volterra

$$\begin{cases} y_1'(t) = 2y_1 - \alpha y_1 y_2 \\ y_2'(t) = -y_2 + \alpha y_1 y_2 \end{cases} \qquad \alpha = 0.01$$

e l'equazione differenziale della caduta di un grave in presenza di resistenza viscosa ρ

$$\frac{dv}{dx} = -g - \rho v$$

```
Lotka-Volterra (RK45, C.I.: [2  3])
69 successful steps
10 failed attempts
475 function evaluations
Lotka-Volterra (RK45, C.I.: [300  150])
42 successful steps
14 failed attempts
337 function evaluations
Lotka-Volterra (RK45, C.I.: [15  22])
57 successful steps
11 failed attempts
409 function evaluations

Caduta libera (RK45, p: 1.5, C.I.: [0])
35 successful steps
2 failed attempts
223 function evaluations
Caduta libera (RK45, p: 0.05, C.I.: [300])
11 successful steps
0 failed attempts
67 function evaluations
```

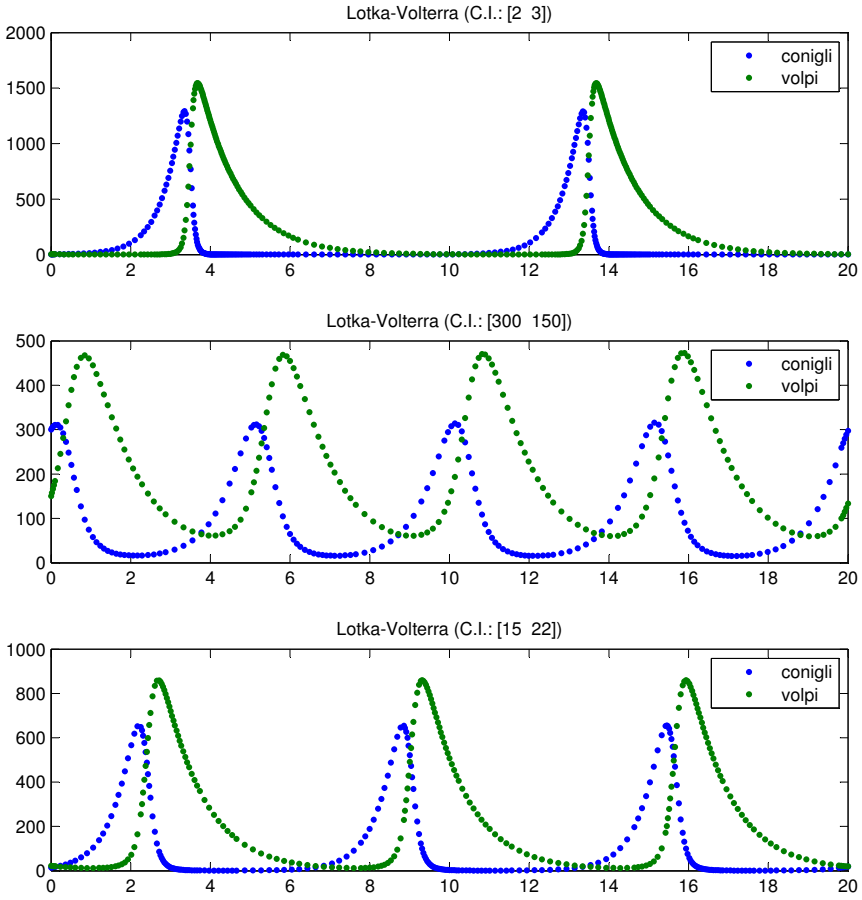

Figura 55: Lotka-Volterra: equazioni del moto. *I predatori prosperano quando c'è abbondanza di prede ma, alla lunga, si ritrovano senza cibo sufficiente per tutti e cominciano ad estinguersi. Mentre la popolazione dei predatori decresce quella delle prede aumenta di nuovo. Questa dinamica continua in un ciclo di crescita e decrescita (di periodo circa 5 unità).*

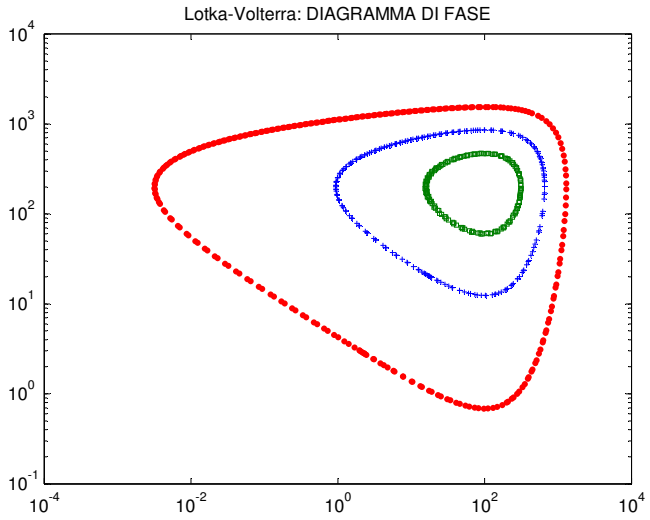

Figura 56: Lotka-Volterra: diagramma di fase

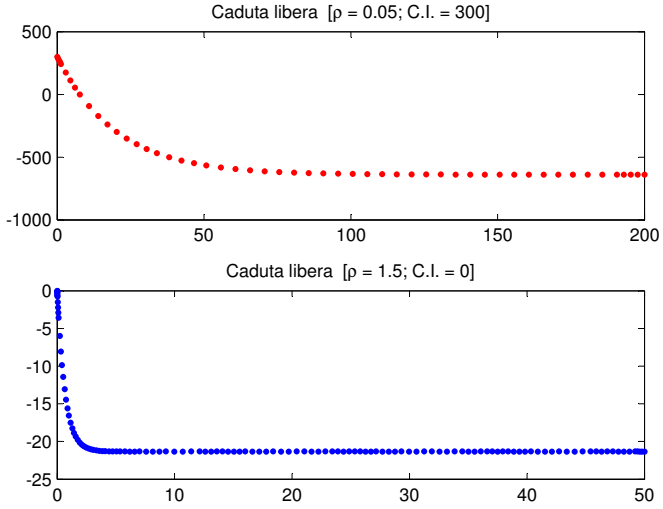

Figura 57: caduta di un grave in presenza di resistenza viscosa. *Sia ρ il coefficiente di viscosità e c_0 una costante che dipende dai dati iniziali, allora* $v(t) = \frac{mg}{\rho}(1 - e^{-\frac{\rho}{m}t - c_0})$. *La costante di tempo $\frac{m}{\rho}$ decade al crescere di ρ; mentre il valore della velocità limite $\frac{mg}{\rho}$ cresce al decrescere di ρ.*

17.2 *esercizio 3*

Risolvere con il solutore `ode23()` e `ode45()` (i.e. solutori che combinano rispettivamente metodi di Runge Kutta di ordine II/III e IV/V; questo è chiaramente più accurato di quello, il quale tuttavia può essere più efficiente per tolleranze lasche e per problemi moderatamente stiffness) l'equazione differenziale

$$y' + \frac{2x}{x^2-1}y = \frac{\cos x}{x^2-1} \quad y(0) = 1 \quad (-1 < x < 1)$$

e confrontare l'errore assoluto con la soluzione esatta

$$y = \frac{\sin x - 1}{x^2 - 1}$$

Nota Nella presente sezione esibiamo soltanto alcuni frammenti di codice e/o ci limitiamo ad esibirne gli output, giacché gli `script` sono essenzialmente gli stessi, salvo piccole correzioni poco significative e in genere di natura grafica.

```
f = inline('(sin(x)-1)/(x^2-1)');
f = vectorize(f);
b = 0.8; y0 = 1;

figure
subplot(2,1,1)
tspan = [0,b]; % intervallo [0,1]
disp(['F3 (ode45; C.I.: [0,',num2str(y0),'])'])
[t1,y1] = ode45('f3',tspan,y0, odeset('stats','on')); % ode45
disp(['F3 (ode23; C.I.: [0,',num2str(y0),'])'])
[t2,y2] = ode23('f3',tspan,y0, odeset('stats','on')); % ode23
err451=abs((y1-f(t1))./f(t1)); t45=t1; % err ass ode45
err231=abs((y2-f(t2))./f(t2)); t23=t2; % err ass ode23
plot(t1,y1,'.','MarkerSize',10)
hold on
plot(t2,y2,'+','MarkerSize',3,'Color',[0 0.498039215803146 0])

tspan = [0,-b]; % intervallo [-1,0]
disp(['F3 (ode45; C.I.: [0,',num2str(y0),'])'])
[t1,y1] = ode45('f3',tspan,y0, odeset('stats','on')); % ode45
disp(['F3 (ode23; C.I.: [0,',num2str(y0),'])'])
[t2,y2] = ode23('f3',tspan,y0, odeset('stats','on')); % ode23
err452=abs((y1-f(t1))./f(t1));  % err ass ode45
err232=abs((y2-f(t2))./f(t2));  % err ass ode23
plot(t1,y1,'.','MarkerSize',10)
plot(t2,y2,'+','MarkerSize',6,'Color',[0 0.498039215803146 0])
title(['ode45 - ode23; C.I.: [0,',num2str(y0),']'])
legend('ode45','ode23')
```

```
xx = linspace(-b,b);
yy = f(xx);
plot(xx,yy,'r','Linewidth',1) % grafico della funzione

subplot(2,1,2)
t45=[t1;t45]; err45=[err452; err451]; % err ass ode45
t23=[t2;t23]; err23=[err232; err231]; % err ass ode23
semilogy(t45,err45,'.','MarkerSize',10)
hold on
semilogy(t23,err23,'Marker','+','MarkerSize',6,
     ...'LineStyle','none','Color',[0 0.498039215803146 0])
title(['errore assoluto ode45 - ode23'])
```

```
F3 (ode45; C.I.: [0,1])
10 successful steps
0 failed attempts
61 function evaluations

F3 (ode23; C.I.: [0,1])
11 successful steps
1 failed attempts
37 function evaluations
```

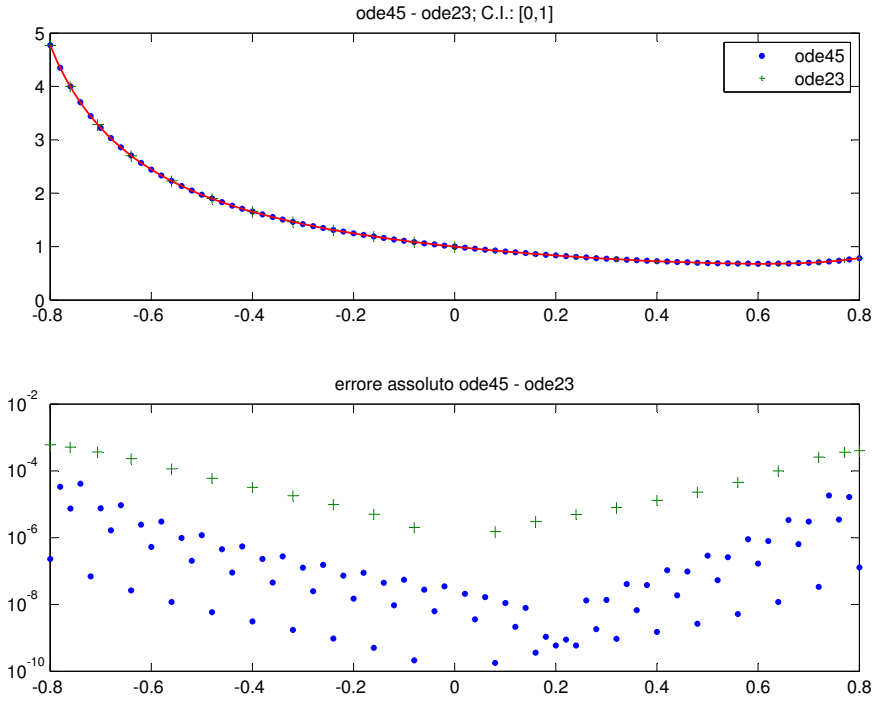

Figura 58: confronto dei solutori `ode23()` e `ode45()` per la risoluzione della seguente equazione differenziale: $y' + \frac{2x}{x^2-1} y = \frac{\cos x}{x^2-1}$.

17.3 *esercizio 4*

Con un opportuno solutore di `matlab` approssimare i seguenti problemi. Giustificare la scelta anche mostrando confronti con altri solutori.
Si definisce problema stiff (secondo Lambert)

> *If a numerical method with a finite region of absolute stability, applied to a system with any initial conditions, is forced to use in a certain interval of integration a steplength which is excessively small in relation to the smoothness of the exact solution in that interval, then the system is said to be stiff in that interval.*

1.

$$\begin{cases} y'(t) = -10^3(y - e^{-t}) - e^{-t} \\ y(0) = 0 \end{cases}$$

```
F4_1 (ode45; C.I.: [0,0])
341 successful steps
21 failed attempts
2173 function evaluations

F4_1 (ode23; C.I.: [0,0])
451 successful steps
4 failed attempts
1366 function evaluations

F4_1 (ode133; C.I.: [0,0])
710 successful steps
97 failed attempts
1518 function evaluations

F4_1 (ode15s; C.I.: [0,0])
48 successful steps
0 failed attempts
64 function evaluations
1 partial derivatives
12 LU decompositions
60 solutions of linear systems

F4_1 (ode23s; C.I.: [0,0])
47 successful steps
0 failed attempts
191 function evaluations
47 partial derivatives
47 LU decompositions
```

```
F4_1 (ode23t; C.I.: [0,0])
47 successful steps
0 failed attempts
76 function evaluations
1 partial derivatives
25 LU decompositions
72 solutions of linear systems

F4_1 (ode23tb; C.I.: [0,0])
34 successful steps
0 failed attempts
100 function evaluations
1 partial derivatives
20 LU decompositions
130 solutions of linear systems
```

2.

$$\begin{cases} y_1'(t) = y_2 y_3 \\ y_2'(t) = -y_1 y_3 \\ y_3'(t) = -0.51 y_2 y_1 \end{cases} \quad \begin{cases} y_1(0) = 0 \\ y_2(0) = 1 \\ y_3(0) = 1 \end{cases}$$

Nota Omettiamo i dettagli, per altro poco significativi, per chiarezza espositiva. La gestione del problema vettoriale non è dissimile dal caso scalare.

3. (Equazione di Van der Pol)

$$\begin{cases} y'' - \mu(1-y^2)y' + y = 0 \\ y(0) = 2 \\ y'(0) = 0 \end{cases} \Leftrightarrow \begin{cases} y_1'(t) = y_2 \\ y_2'(t) = \mu(1-y_1^2)y_2 - y_1 \\ y_1(0) = 2 \\ y_2(0) = 0 \end{cases}$$

Confrontiamo i valori per $\mu = 1$ su $[0, 20]$ e $\mu = 1000$ su $[0, 3000]$

```
F4_3 (ode45; C.I.: [0, 2  0])
59 successful steps - 415 function evaluations
F4_3 (ode23; C.I.: [0, 2  0])
161 successful steps - 547 function evaluations
F4_3 (ode133; C.I.: [0, 2  0])
175 successful steps - 363 function evaluations
F4_3 (ode15s; C.I.: [0, 2  0])
208 successful steps - 448 function evaluations
F4_3 (ode23s; C.I.: [0, 2  0])
171 successful steps - 903 function evaluations
```

```
F4_3 (ode23t; C.I.: [0, 2  0])
297 successful steps - 570 function evaluations
F4_3 (ode23tb; C.I.: [0, 2  0])
227 successful steps - 846 function evaluations

            ***

F4_4 (ode45; C.I.: [0, 2  0])
1.68411e+006 successful steps
112240 failed attempts
1.07781e+007 function evaluations

F4_4 (ode23; C.I.: [0, 2  0])
2.2245e+006 successful steps
39 failed attempts
6.67361e+006 function evaluations

F4_4 (ode133; C.I.: [0, 2  0])
3.47754e+006 successful steps
475583 failed attempts
7.43066e+006 function evaluations

F4_4 (ode15s; C.I.: [0, 2  0])
591 successful steps
225 failed attempts
1883 function evaluations
45 partial derivatives
289 LU decompositions
1747 solutions of linear systems

F4_4 (ode23s; C.I.: [0, 2  0])
743 successful steps
17 failed attempts
3751 function evaluations
743 partial derivatives
760 LU decompositions
2280 solutions of linear systems

F4_4 (ode23t; C.I.: [0, 2  0])
776 successful steps
94 failed attempts
2121 function evaluations
36 partial derivatives
294 LU decompositions
2012 solutions of linear systems
```

```
F4_4 (ode23tb; C.I.: [0, 2  0])
573 successful steps
93 failed attempts
2947 function evaluations
44 partial derivatives
269 LU decompositions
3415 solutions of linear systems
```

Osservazioni

$f_1(x)$

$$y'(t) = -10^3(y - e^{-t}) - e^{-t} = -10^3 y - \phi(t)$$

Poiché la regione di assoluta stabilità del metodo è finita e la sua traccia reale è contenuta nell'intervallo $(-3, 0)$, stimiamo la scelta del massimo passo stabile $h_{max} = 0.003$. La soluzione del problema di Cauchy presenta un piccolo transiente che impone un eccessivo raffinamento del passo di integrazione:

$$y(t) = e^{-t} + e^{-10^3 t}$$

ode45: 341 successful steps - 2173 function evaluations
ode15s: 48 successful steps - 64 function evaluations

$f_4(x)$

$$\begin{cases} y'' - \mu(1-y^2)y' + y = 0 \\ y(0) = 2 \\ y'(0) = 0 \end{cases} \Leftrightarrow \begin{cases} y_1'(t) = y_2 \\ y_2'(t) = \mu y_2 - y_1 + o(|(y_1, y_2)|) \\ y_1(0) = 2 \\ y_2(0) = 0 \end{cases}$$

$$\begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & \mu \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + o\left(\left\| \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \right\| \right)$$

$$p(x) = det(A - x) = det \begin{bmatrix} -x & 1 \\ -1 & \mu - x \end{bmatrix} = x^2 - \mu x + 1 = 0$$

$$r_s = \frac{\lambda_{max}}{\lambda_{min}} \sim \mu^2 \to +\infty \qquad \mu \to +\infty$$

Pertanto, per $\mu = 10^3$, $r_s = 10^6$. Tuttavia, secondo la definizione non possiamo definire stiff il presente problema di Cauchy, poiché gli autovalori della matrice Iacobiana hanno parte reale positiva. Tuttavia, per μ grande, i metodi stiff risultano più efficienti di Runge Kutta espliciti.

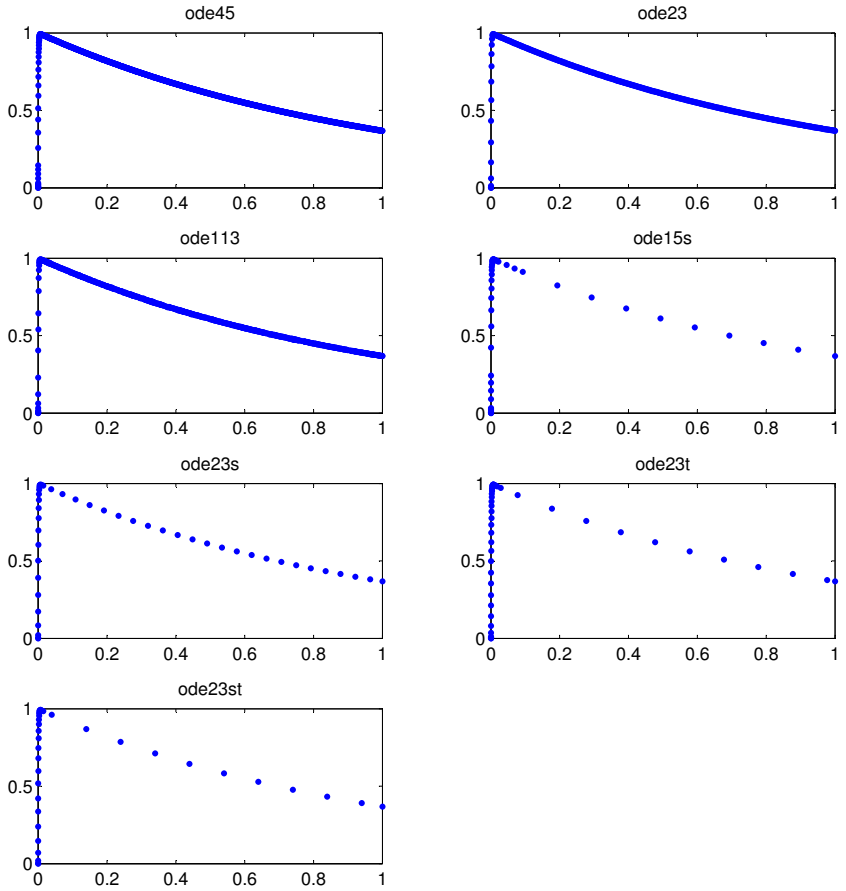

Figura 59: confronto dei solutori di `matlab` per la risoluzione della seguente equazione differenziale: $y'(t) = -10^3(y - e^{-t}) - e^{-t}$

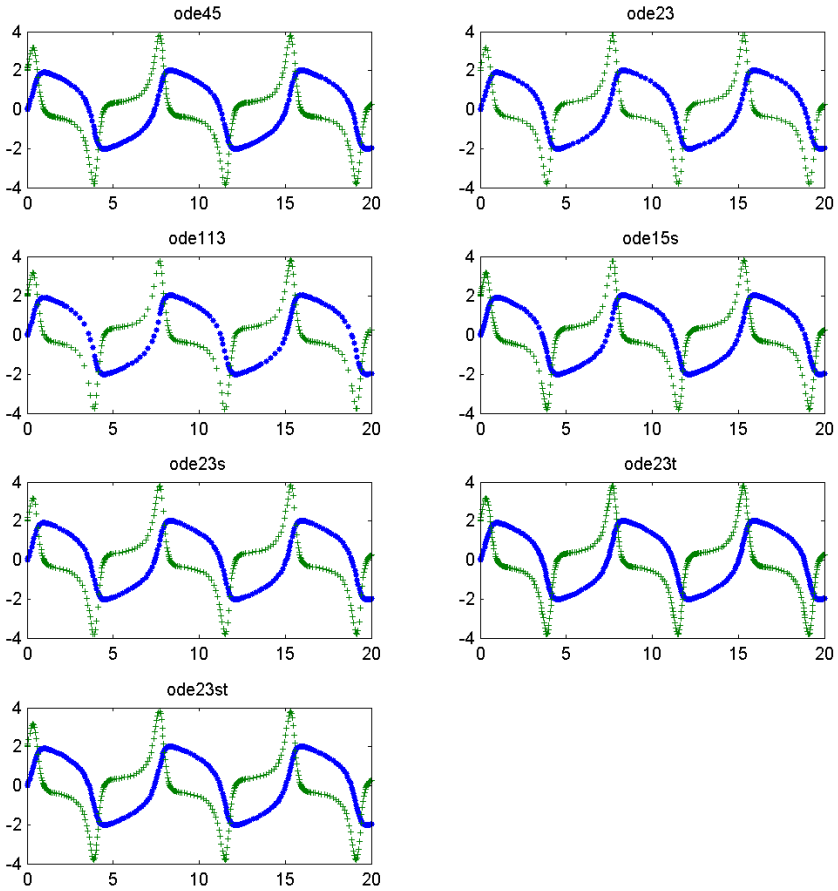

Figura 60: confronto dei solutori di `matlab` per la risoluzione della seguente equazione differenziale (equazione di Van der Pol): $y'' - \mu(1-y^2)y' + y = 0$ $[\mu = 1]$

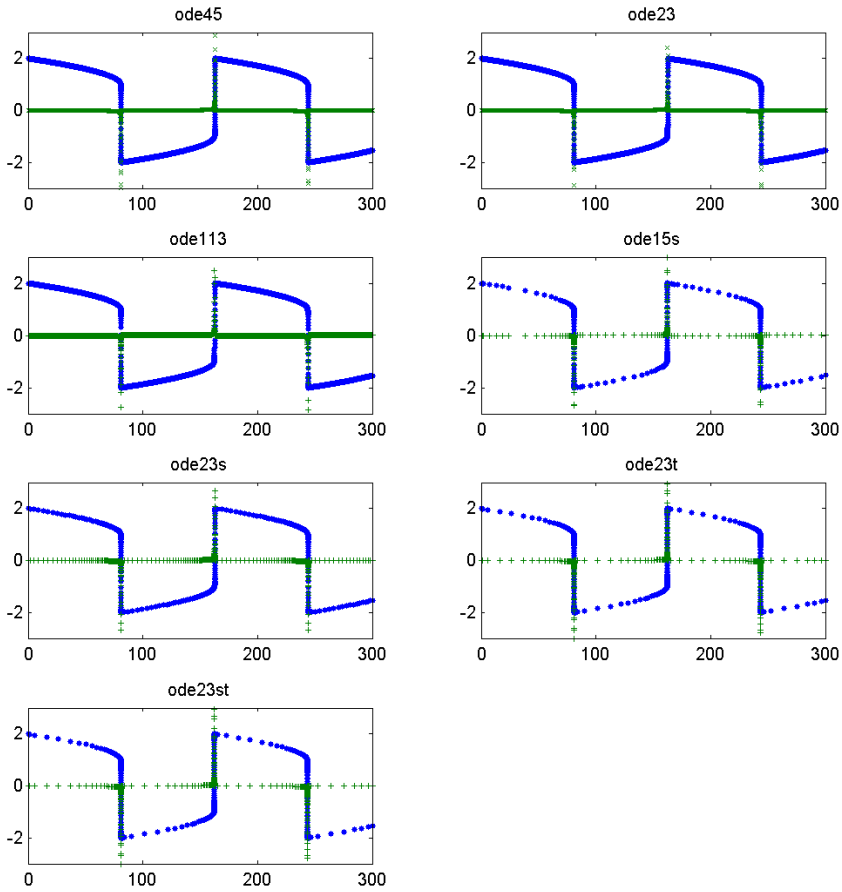

Figura 61: confronto dei solutori di `matlab` per la risoluzione della seguente equazione differenziale (equazione di Van der Pol): $y'' - \mu(1-y^2)y' + y = 0$ $[\mu = 1000]$. *I metodi Runge Kutta impliciti (in generale con regione di assoluta stabilità illimitata) sono più stabili dei metodi Runge Kutta espliciti.*

ode45:	1.68411e6 successful steps - 1.07781e7 function evaluations
ode23:	2.2245e6 successful steps - 6.67361e6 function evaluations
ode133:	3.47754e6 successful steps - 7.43066e6 function evaluations
ode15s:	591 successful steps - 1883 function evaluations
ode23s:	743 successful steps - 3751 function evaluations
ode23t:	776 successful steps - 2121 function evaluations
ode23st:	573 successful steps - 2947 function evaluations

Conclusioni

I metodi Runge Kutta espliciti non sono sufficientemente stabili per l'integrazione di un sistema stiff: per avere risultati accettabili è necessario utilizzare un passo di integrazione troppo piccolo per assecondare il forte decadimento del transiente. In corrispondenza di un rapporto di stiffness elevato (i.e. il rapporto tra gli autovalori massimo e minimo di un sistema lineari di equazioni differenziali ordinarie del prim'ordine[2]), si impone la scelta di un passo di integrazione h molto piccolo, affinché il prodotto λh, ove per funzione test sia stata scelta $\dot{x} = \lambda x$ (nel caso vettoriale si ponga λ l'autovalore di modulo massimo), appartenga alla regione di assoluta stabilità. Di qui la scelta di metodi (e.g. impliciti) con una grande (illimitata) regione di assoluta stabilità.

[2]La stabilità di un assegnato sistema non lineare di equazioni differenziali ordinarie (autonomo in forma normale), $\dot{x} = f(x)$, nel punto $x = 0$, coincide almeno localmente (nel senso dei teoremi di Lyapunov) con la stabilità del suo linearizzato, i.e. il sistema lineare associato alla matrice Iacobiana di f

$$\dot{x}_j = \sum_{k=1}^{n} \frac{\partial f_j}{\partial x}(0) x_j$$

La fondatezza della procedura di linearizzazione al fine di determinare la stabilità del sistema originario è riposta in questa osservazione.

Elenco delle figure